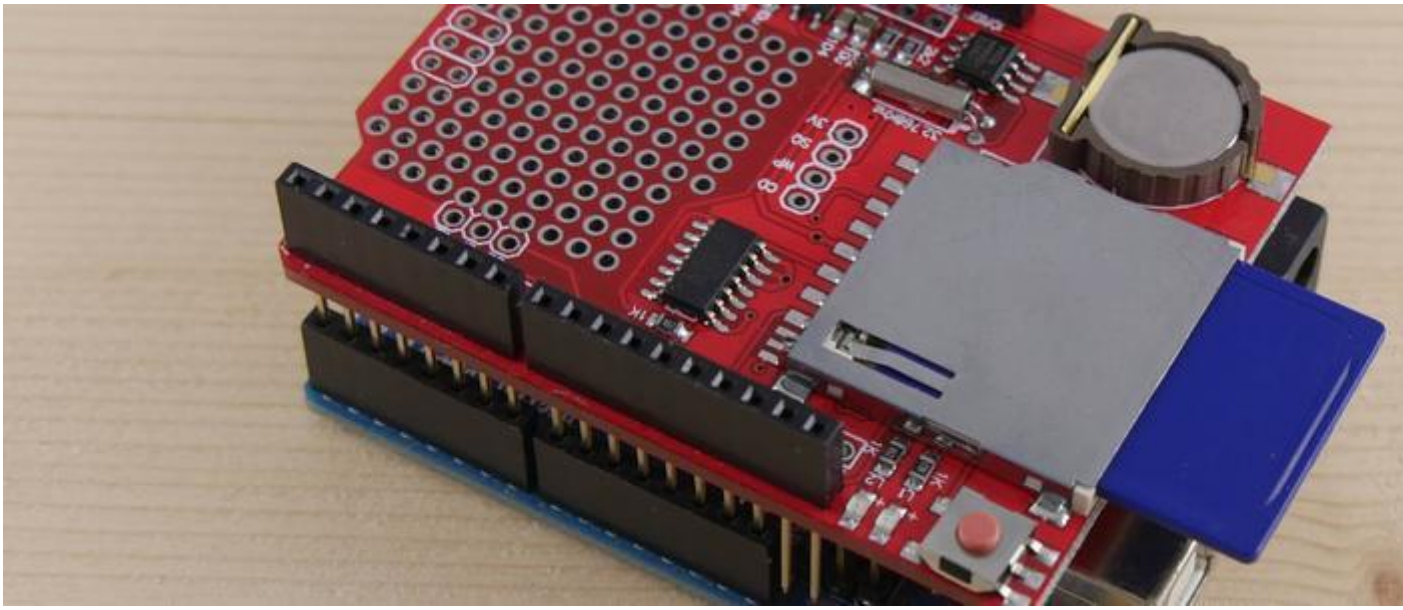
Fabriquer un système d'acquisition de mesures (datalogger) avec une carte Arduino / Genuino

Avec ou sans horodatage, c'est vous qui choisissez

https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/



par skywodd | fév. 15, 2017 | Licence (voir pied de page)

Catégories : Tutoriels Arduino | Mots clefs : Arduino Genuino Datalogger Mesure DS1307 RTC Carte SD SD card

Cet article a été modifié pour la dernière fois le mars 14, 2017 à 5:49 après-midi

Cet article n'a pas été mis à jour depuis un certain temps, son contenu n'est peut être plus d'actualité.

Dans ce tutoriel, nous allons fabriquer un système d'acquisition de mesures autonome (aka "datalogger"). Celui-ci stockera les mesures sur une carte SD dans un format

textuel facilement utilisable avec un logiciel de type tableur (Excel, Libre Calc, etc.). Nous étudierons deux versions du code : une version basique sans horodatage pour comprendre les bases du fonctionnement et une version plus complexe avec horodatage pour obtenir des mesures utilisables par la suite.

Sommaire

- Le principe d'un datalogger
- Le montage
- Le code de base
- Le code avec horodatage
- Conclusion

Bonjour à toutes et à tous !

Dans un précédent article, nous avons vu ensemble comment utiliser une carte SD avec une carte Arduino. Aujourd'hui, nous allons mettre en oeuvre ces nouvelles connaissances dans le but de construire un système de relevé de mesures autonome sur carte SD.

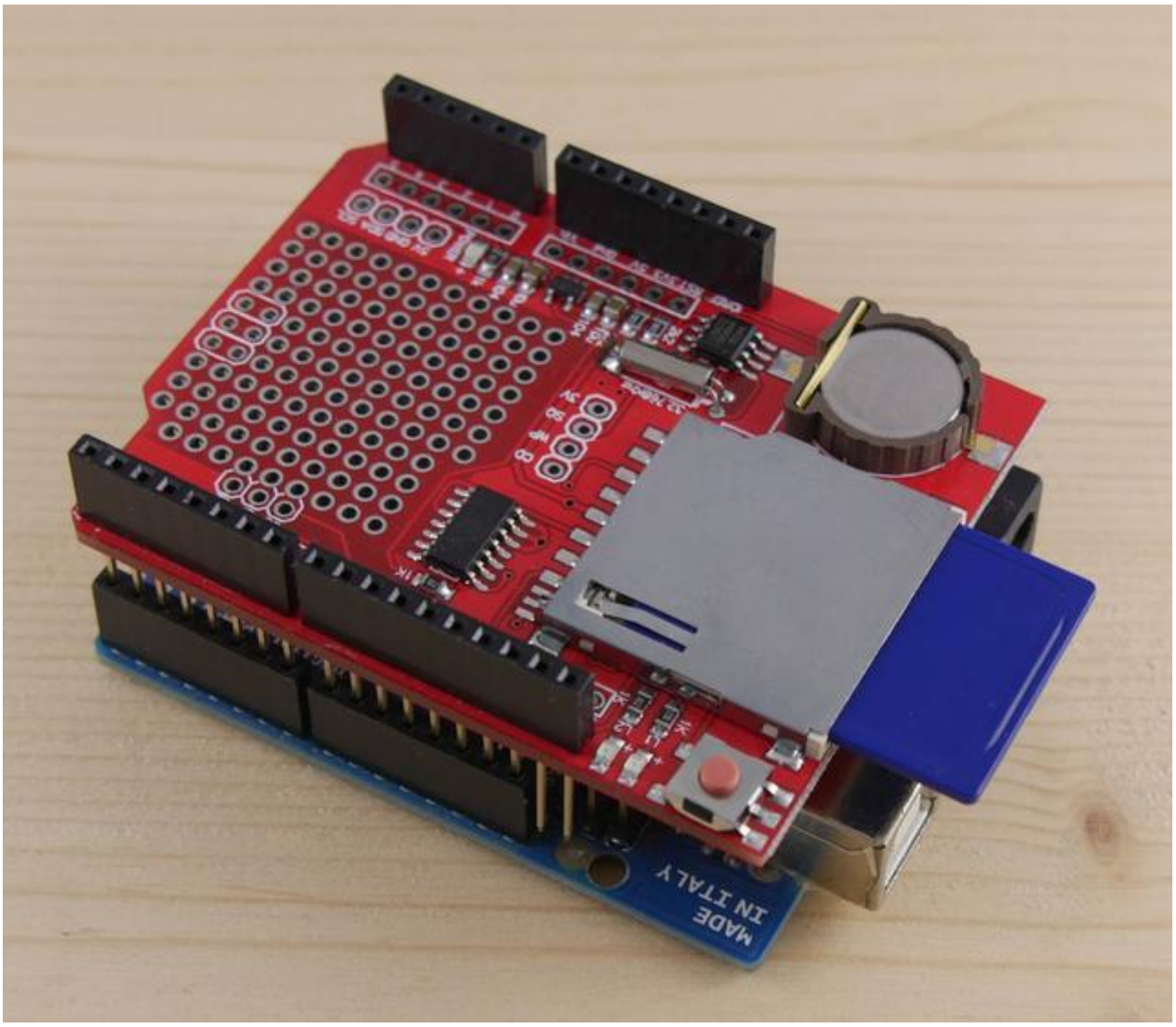
Le principe d'un datalogger

Un datalogger ("enregistreur de données", ou "système d'acquisition de données" en bon français) est un type de montage un peu particulier n'ayant qu'un seul et unique but : enregistrer des données de manière périodique.

Le principe est simple : toutes les N secondes, le système fait une mesure, enregistre ladite mesure dans une mémoire non volatile comme une carte SD, une mémoire EEPROM, ou tout simplement sur une feuille de papier et … voilà.

Ce genre de montage est utilisé dans diverses situations. Si vous vous demandez comment évolue la température dans une pièce de votre maison par exemple, vous pouvez mettre en oeuvre un datalogger pour voir l'évolution de la température sur une période de temps donnée. Si vous êtes un météorologue, vous pouvez utiliser le même type d'outil pour mesurer l'hygrométrie, la pression atmosphérique, la température, etc.

Le montage



Datalogger minimaliste

Dans le cadre de ce tutoriel, nous allons avoir besoin de deux choses :

- Une carte Arduino UNO (et son câble USB),
- Une shield SD avec un module RTC DS1307 (pour le prochain chapitre).

Le montage se limite à empiler les deux cartes l'une sur l'autre.

PS Si vous utilisez un module SD et un module DS1307 séparés, je vous invite à lire mes deux articles sur chacun de ses sujets ici et ici pour plus de détails.

Et les capteurs ?

Comme il existe autant de capteurs différents disponibles dans le commerce que de grains de sable dans le désert du Sahara, je ne vais pas utiliser de capteur dans ce tutoriel. Ce tutoriel

sera une base de travail que vous pourrez personnaliser à votre guise, plutôt qu'un tutoriel spécialisé sur un capteur en particulier.

J'utilise les entrées analogiques de la carte Arduino en guise de capteur imaginaire. À vous de remplacer la partie mesure dans le code des chapitres suivant par votre code de mesure 

N.B. Les broches A4 etA5 seront utilisées par le bus I2C pour le module d'horodatage. Ne connectez rien sur ces broches à part des modules I2C.

Le code de base

Commençons simplement avec un code qui écrit périodiquement des mesures brutes dans un fichier.

```
1 /* Dépendances */
2 #include <SPI.h> // Pour la communication SPI
3 #include <SD.h>  // Pour la communication avec la carte SD
```

Tout d'abord, nous allons avoir besoin de deux bibliothèques standards fournies avec le logiciel Arduino : SPI et SD. Ces deux bibliothèques permettent ensemble de communiquer avec la carte SD.

```
1 /** Broche CS de la carte SD */
2 const byte SDCARD_CS_PIN = 10; // A remplacer suivant votre shield SD
3
4 /** Nom du fichier de sortie */
5 const char* OUTPUT_FILENAME = "data.csv";
6
7 /** Delai entre deux prises de mesures */
8 const unsigned long DELAY_BETWEEN_MEASURES = 5000;
```

Viennent ensuite les classiques constantes.

Dans ce programme, nous allons avoir besoin de trois constantes : le numéro de broche `CS` de la carte SD, le nom du fichier de sortie qui contiendra nos données et le délai en millisecondes entre deux mesures.

```
1 /** Fichier de sortie avec les mesures */
2 File file;
```

Exceptionnellement, nous allons avoir besoin d'une variable globale dans ce tutoriel.

Habituellement, je conseille d'éviter au maximum les variables globales, car celles-ci sont souvent le signe d'une mauvaise organisation dans son code. Ici, pour une fois, on va avoir une utilisation légitime d'une variable globale (accessible partout dans le code).

Cette variable globale sera l'objet `File` qui va nous permettre d'écrire des données dans le fichier de sortie.

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

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

  /* Initialisation du port SPI */
  pinMode(10, OUTPUT); // Arduino UNO
  //pinMode(53, OUTPUT); // Arduino Mega

  /* Initialisation de la carte SD */
  Serial.println(F("Initialisation de la carte SD ... "));
  if (!SD.begin(SDCARD_CS_PIN)) {
    Serial.println(F("Erreur : Impossible d'initialiser la carte SD"));
    Serial.println(F("Verifiez la carte SD et appuyez sur le bouton RESET"));
    for (;;); // Attend appui sur bouton RESET
  }

  /* Ouvre le fichier de sortie en écriture */
  Serial.println(F("Ouverture du fichier de sortie ... "));
  file = SD.open(OUTPUT_FILENAME, FILE_WRITE);
  if (!file) {
    Serial.println(F("Erreur : Impossible d'ouvrir le fichier de sortie"));
    Serial.println(F("Verifiez la carte SD et appuyez sur le bouton RESET"));
    for (;;); // Attend appui sur bouton RESET
  }

  /* Ajoute l'entête CSV si le fichier est vide */
  if (file.size() == 0) {
    Serial.println(F("Ecriture de l'entete CSV ..."));
    file.println(F("Tension A0; Tension A1; Tension A2; Tension A3"));
    file.flush();
  }
}
```

La fonction `setup()` fait plein de choses, pas d'inquiétude, je vais détailler tout cela 

- Tout d'abord la fonction `setup()` initialise le port série pour avoir un retour sur le moniteur série.
- La fonction `setup()` initialise ensuite la broche `D10` (`D53` sur les cartes Mega) pour le port SPI et initialise la carte SD dans la foulée.

  Si une erreur survient durant l'initialisation de la carte SD, on affiche un message d'erreur et on attend le redémarrage de la carte Arduino.

- Une fois la carte SD initialisé, la fonction `setup()` ouvre en écriture le fichier de sortie. Si le fichier n'existe pas, il est automatiquement créé.

  Si pour une quelconque raison l'ouverture du fichier échoue, on affiche un message d'erreur et on bloque.

- Pour terminer, la fonction `setup()` écrit dans le fichier une ligne de texte au format CSV. Cette ligne de texte est un entête qui permettra plus tard d'importer les données dans un logiciel comme Excel ou LibreOffice Calc.

  Sans entrer dans les détails du format CSV (cf lien Wikipedia ci-dessus), une ligne de texte équivaut à une prise de mesure. Chaque ligne peut contenir plusieurs champs / mesures en séparant chaque champ par un séparateur, un simple point virgule dans mon cas (vous pouvez choisir autre chose si vous le souhaitez).

```
/** Fonction loop() */
void loop() {
  // Temps de la précédente mesure et actuel
  static unsigned long previousMillis = 0;
  unsigned long currentMillis = millis();

  /* Réalise une prise de mesure toutes les DELAY_BETWEEN_MEASURES
millisecondes */
  if (currentMillis - previousMillis >= DELAY_BETWEEN_MEASURES) {
    previousMillis = currentMillis;
    measure();
  }
}
```

La fonction `loop()` vous semble peut-être familière. J'ai repris le code de mon exemple de multitâche non bloquant pour cet article.

Ce que font ces quelques lignes de code est assez simple : appeler la fonction `measure()` toutes les `DELAY_BETWEEN_MEASURES` millisecondes. Rien de plus.

```
/** Fonction de mesure - à personnaliser selon ses besoins */
void measure() {

  /* Réalise la mesure */
  float v_1 = analogRead(A0) * 5.0 / 1023;
  float v_2 = analogRead(A1) * 5.0 / 1023;
  float v_3 = analogRead(A2) * 5.0 / 1023;
  float v_4 = analogRead(A3) * 5.0 / 1023;

  /* Affiche les données sur le port série pour debug */
  Serial.print(v_1);
  Serial.print(F("; "));
  Serial.print(v_2);
  Serial.print(F("; "));
```

```
15   Serial.print(v_3);
16   Serial.print(F("; "));
17   Serial.println(v_4);
18
19   /* Enregistre les données sur la carte SD */
20   file.print(v_1);
21   file.print(F("; "));
22   file.print(v_2);
23   file.print(F("; "));
24   file.print(v_3);
25   file.print(F("; "));
26   file.println(v_4);
27   file.flush();
28 }
```

La fonction `measure()` fait le plus gros du travail. Elle fait la prise de mesures et écrit les données sur la carte SD.

Astuce : Pensez à afficher les mesures sur le port série avant des les écrites sur la carte SD. Cela vous aidera durant les tests en vous permettant de visualiser les données en temps réels dans le moniteur série. ***N.B. Ne pas oublier d'appeler `file.flush()` après avoir écrit les données de la prise de mesures. Si vous oubliez ce détail, vous risquez de perdre des données en cas de problème d'alimentation ou de redémarrage de la carte Arduino. Le code complet avec commentaires :***

```
 1 /**
 2  * Exemple de code Arduino pour un datalogger basique avec stockage sur
 3 carte SD.
 4  */
 5
 6 /* Dépendances */
 7 #include <SPI.h> // Pour la communication SPI
 8 #include <SD.h>  // Pour la communication avec la carte SD
 9
10
11 /** Broche CS de la carte SD */
12 const byte SDCARD_CS_PIN = 10; // A remplacer suivant votre shield SD
13
14 /** Nom du fichier de sortie */
15 const char* OUTPUT_FILENAME = "data.csv";
16
17 /** Delai entre deux prise de mesures */
18 const unsigned long DELAY_BETWEEN_MEASURES = 5000;
19
20
21 /** Fichier de sortie avec les mesures */
22 File file;
23
```

```
24
25 /** Fonction setup() */
26 void setup() {
27
28   /* Initialisation du port série (debug) */
29   Serial.begin(115200);
30
31   /* Initialisation du port SPI */
32   pinMode(10, OUTPUT); // Arduino UNO
33   //pinMode(53, OUTPUT); // Arduino Mega
34
35   /* Initialisation de la carte SD */
36   Serial.println(F("Initialisation de la carte SD ... "));
37   if (!SD.begin(SDCARD_CS_PIN)) {
38     Serial.println(F("Erreur : Impossible d'initialiser la carte SD"));
39     Serial.println(F("Verifiez la carte SD et appuyez sur le bouton
40 RESET"));
41     for (;;); // Attend appui sur bouton RESET
42   }
43
44   /* Ouvre le fichier de sortie en écriture */
45   Serial.println(F("Ouverture du fichier de sortie ... "));
46   file = SD.open(OUTPUT_FILENAME, FILE_WRITE);
47   if (!file) {
48     Serial.println(F("Erreur : Impossible d'ouvrir le fichier de
49 sortie"));
50     Serial.println(F("Verifiez la carte SD et appuyez sur le bouton
51 RESET"));
52     for (;;); // Attend appui sur bouton RESET
53   }
54
55   /* Ajoute l'entête CSV si le fichier est vide */
56   if (file.size() == 0) {
57     Serial.println(F("Ecriture de l'entete CSV ..."));
58     file.println(F("Tension A0; Tension A1; Tension A2; Tension A3"));
59     file.flush();
60   }
61 }
62
63
64 /** Fonction loop() */
65 void loop() {
66   // Temps de la précédente mesure et actuel
67   static unsigned long previousMillis = 0;
68   unsigned long currentMillis = millis();
69
70   /* Réalise une prise de mesure toutes les DELAY_BETWEEN_MEASURES
71 millisecondes */
72   if (currentMillis - previousMillis >= DELAY_BETWEEN_MEASURES) {
```

```
    previousMillis = currentMillis;
    measure();
  }
}


/** Fonction de mesure - à personnaliser selon ses besoins */
void measure() {

  /* Réalise la mesure */
  float v_1 = analogRead(A0) * 5.0 / 1023;
  float v_2 = analogRead(A1) * 5.0 / 1023;
  float v_3 = analogRead(A2) * 5.0 / 1023;
  float v_4 = analogRead(A3) * 5.0 / 1023;

  /* Affiche les données sur le port série pour debug */
  Serial.print(v_1);
  Serial.print(F("; "));
  Serial.print(v_2);
  Serial.print(F("; "));
  Serial.print(v_3);
  Serial.print(F("; "));
  Serial.println(v_4);

  /* Enregistre les données sur la carte SD */
  file.print(v_1);
  file.print(F("; "));
  file.print(v_2);
  file.print(F("; "));
  file.print(v_3);
  file.print(F("; "));
  file.println(v_4);
  file.flush();
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Le code avec horodatage

Dans le précédent chapitre, on a vu les bases du code du datalogger. Cependant, il manque quelque chose d'absolument nécessaire à notre système : l'horodatage des mesures.

Sans horodatage, impossible de savoir de quand datent les données dans le fichier. Il peut très bien y avoir un "trou" de plusieurs heures, cela est impossible à détecter sans une indication de l'heure de la prise de mesures.

On pourrait utiliser la fonction `millis()` pour connaitre le temps depuis le démarrage de la carte Arduino. Seulement, en cas de redémarrage de la carte Arduino, `millis()` repart de

zéro. On se retrouverait donc avec des données inexploitables, faute de pouvoir savoir à quelle heure une mesure en particulier a été prise.

On va donc mettre en oeuvre un module horloge temps réel DS1307, afin d'avoir l'heure exacte de chaque prise de mesures.

N.B. L'utilisation d'un module DS1307 avec une carte Arduino a fait l'objet d' un article dédié que je vous recommande 

```
/* Dépendances */
#include <Wire.h> // Pour la communication I2C
#include <SPI.h>  // Pour la communication SPI
#include <SD.h>   // Pour la communication avec la carte SD
#include "DS1307.h" // Pour le module DS1307


/* Rétro-compatibilité avec Arduino 1.x et antérieur */
#if ARDUINO >= 100
#define Wire_write(x) Wire.write(x)
#define Wire_read() Wire.read()
#else
#define Wire_write(x) Wire.send(x)
#define Wire_read() Wire.receive()
#endif
```

Afin de pouvoir utiliser le module horloge, il va falloir ajouter deux `#include` à notre code : un pour la bibliothèque Wire qui permet de communiquer avec le module horloge et un second pour inclure le fichier `DS1307.h`, disponible dans mon article dédié en lien un peu plus haut.

On ajoute aussi un bloc de code / magie noire permettant de rendre le code compatible avec les anciennes versions du logiciel Arduino.

N.B. Le fichier `DS1307.h` devra être au même niveau que le fichier de code Arduino. L'article sur le module DS1307 explique cela en détail.

```
/** Fonction de conversion BCD -> decimal */
byte bcd_to_decimal(byte bcd) {
  return (bcd / 16 * 10) + (bcd % 16);
}

/**
 * Fonction récupérant l'heure et la date courante à partir du module
RTC.
 * Place les valeurs lues dans la structure passée en argument (par
pointeur).
 * N.B. Retourne 1 si le module RTC est arrêté (plus de batterie, horloge
arrêtée manuellement, etc.), 0 le reste du temps.
```

```
 */
byte read_current_datetime(DateTime_t *datetime) {

  /* Début de la transaction I2C */
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire_write((byte) 0); // Lecture mémoire à l'adresse 0x00
  Wire.endTransmission(); // Fin de la transaction I2C

  /* Lit 7 octets depuis la mémoire du module RTC */
  Wire.requestFrom(DS1307_ADDRESS, (byte) 7);
  byte raw_seconds = Wire_read();
  datetime->seconds = bcd_to_decimal(raw_seconds);
  datetime->minutes = bcd_to_decimal(Wire_read());
  byte raw_hours = Wire_read();
  if (raw_hours & 64) { // Format 12h
    datetime->hours = bcd_to_decimal(raw_hours & 31);
    datetime->is_pm = raw_hours & 32;
  } else { // Format 24h
    datetime->hours = bcd_to_decimal(raw_hours & 63);
    datetime->is_pm = 0;
  }
  datetime->day_of_week = bcd_to_decimal(Wire_read());
  datetime->days = bcd_to_decimal(Wire_read());
  datetime->months = bcd_to_decimal(Wire_read());
  datetime->year = bcd_to_decimal(Wire_read());

  /* Si le bit 7 des secondes == 1 : le module RTC est arrêté */
  return raw_seconds & 128;
}
```

Les constantes restent là où elles sont et on vient ajouter deux fonctions pour la partie horloge. Ces fonctions sont expliquées en détail dans mon article dédié au module DS1307.

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

 /* Initialise le port I2C */
 Wire.begin();

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

 /* Vérifie si le module RTC est initialisé */
 DateTime_t now;
 if (read_current_datetime(&now)) {
   Serial.println(F("Erreur : L'horloge du module RTC n'est pas
active"));
   Serial.println(F("Configurer le module RTC avant d'utiliser ce
```

```
programme"));
    for (;;); // Attend appui sur bouton RESET
 }

  // ... la suite du code de setup()

  /* Ajoute l'entête CSV si le fichier est vide */
  if (file.size() == 0) {
    Serial.println(F("Ecriture de l'entete CSV ..."));
    file.println(F("Horodatage; Tension A0; Tension A1; Tension A2;
Tension A3"));
    file.flush();
  }
}
```

Dans la fonction `setup()` on va ajouter une ligne de code pour initialiser le port I2C qui nous permettra de communiquer avec le module horloge et un bloc de code testant l'état de l'horloge en essayant de faire une lecture de l'heure.

Il est important de tester l'état du module horloge dans `setup()`. Si l'horloge n'est pas configurée, cela ne sert à rien d'autoriser la prise de mesures !

Le reste du code de la fonction `setup()` reste inchangé. On ajoute juste la colonne "Horodatage" dans l'entête du fichier CSV. La fonction `loop()` n'est pas modifiée.

```
/** Fonction de mesure - à personnaliser selon ses besoins */
void measure() {

  /* Lit la date et heure courante */
  DateTime_t now;
  read_current_datetime(&now);

  /* Réalise la mesure */
  float v_1 = analogRead(A0) * 5.0 / 1023;
  float v_2 = analogRead(A1) * 5.0 / 1023;
  float v_3 = analogRead(A2) * 5.0 / 1023;
  float v_4 = analogRead(A3) * 5.0 / 1023;

  /* Affiche les données sur le port série pour debug */
  Serial.print(v_1);
  Serial.print(F("; "));
  Serial.print(v_2);
  Serial.print(F("; "));
  Serial.print(v_3);
  Serial.print(F("; "));
  Serial.println(v_4);

  /* Enregistre les données sur la carte SD */
```

```
  // Horodatage
  file.print(now.days);
  file.print(F("/"));
  file.print(now.months);
  file.print(F("/"));
  file.print(now.year + 2000);
  file.print(F(" "));
  file.print(now.hours);
  file.print(F(":"));
  file.print(now.minutes);
  file.print(F(":"));
  file.print(now.seconds);
  file.print(F("; "));
  // Données brutes
  file.print(v_1);
  file.print(F("; "));
  file.print(v_2);
  file.print(F("; "));
  file.print(v_3);
  file.print(F("; "));
  file.println(v_4);
  file.flush();
}
```

La fonction `measure()` se paye un petit coup de lifting.

Désormais on commence par lire l'heure (et la date), puis on fait les mesures. Ensuite on écrit le tout sur la carte SD.

PS Inutile d'afficher l'heure et la date sur le port série. Celui-ci n'est là que pour faire du debug.

Le code complet avec commentaires :

```
/**
 * Exemple de code Arduino pour un datalogger avec horodatage et
stockage sur carte SD.
 */

/* Dépendances */
#include <Wire.h> // Pour la communication I2C
#include <SPI.h>  // Pour la communication SPI
#include <SD.h>   // Pour la communication avec la carte SD
#include "DS1307.h" // Pour le module DS1307


/* Rétro-compatibilité avec Arduino 1.x et antérieur */
#if ARDUINO >= 100
```

```
15 #define Wire_write(x) Wire.write(x)
16 #define Wire_read() Wire.read()
17 #else
18 #define Wire_write(x) Wire.send(x)
19 #define Wire_read() Wire.receive()
20 #endif
21
22
23 /** Broche CS de la carte SD */
24 const byte SDCARD_CS_PIN = 10; // A remplacer suivant votre shield SD
25
26 /** Nom du fichier de sortie */
27 const char* OUTPUT_FILENAME = "data.csv";
28
29 /** Delai entre deux prises de mesures */
30 const unsigned long DELAY_BETWEEN_MEASURES = 5000;
31
32
33 /** Fonction de conversion BCD -> decimal */
34 byte bcd_to_decimal(byte bcd) {
35   return (bcd / 16 * 10) + (bcd % 16);
36 }
37
38 /**
39  * Fonction récupérant l'heure et la date courante à partir du module
40 RTC.
41  * Place les valeurs lues dans la structure passée en argument (par
42 pointeur).
43  * N.B. Retourne 1 si le module RTC est arrêté (plus de batterie,
44 horloge arrêtée manuellement, etc.), 0 le reste du temps.
45  */
46 byte read_current_datetime(DateTime_t *datetime) {
47
48   /* Début de la transaction I2C */
49   Wire.beginTransmission(DS1307_ADDRESS);
50   Wire_write((byte) 0); // Lecture mémoire à l'adresse 0x00
51   Wire.endTransmission(); // Fin de la transaction I2C
52
53   /* Lit 7 octets depuis la mémoire du module RTC */
54   Wire.requestFrom(DS1307_ADDRESS, (byte) 7);
55   byte raw_seconds = Wire_read();
56   datetime->seconds = bcd_to_decimal(raw_seconds);
57   datetime->minutes = bcd_to_decimal(Wire_read());
58   byte raw_hours = Wire_read();
59   if (raw_hours & 64) { // Format 12h
60     datetime->hours = bcd_to_decimal(raw_hours & 31);
61     datetime->is_pm = raw_hours & 32;
62   } else { // Format 24h
63     datetime->hours = bcd_to_decimal(raw_hours & 63);
```

```
64      datetime->is_pm = 0;
65    }
66    datetime->day_of_week = bcd_to_decimal(Wire_read());
67    datetime->days = bcd_to_decimal(Wire_read());
68    datetime->months = bcd_to_decimal(Wire_read());
69    datetime->year = bcd_to_decimal(Wire_read());
70
71    /* Si le bit 7 des secondes == 1 : le module RTC est arrêté */
72    return raw_seconds & 128;
73  }
74
75
76  /** Fichier de sortie avec les mesures */
77  File file;
78
79
80  /** Fonction setup() */
81  void setup() {
82
83    /* Initialise le port I2C */
84    Wire.begin();
85
86    /* Initialisation du port série (debug) */
87    Serial.begin(115200);
88
89    /* Vérifie si le module RTC est initialisé */
90    DateTime_t now;
91    if (read_current_datetime(&now)) {
92      Serial.println(F("Erreur : L'horloge du module RTC n'est pas
93  active"));
94      Serial.println(F("Configurer le module RTC avant d'utiliser ce
95  programme"));
96      for (;;); // Attend appui sur bouton RESET
97    }
98
99    /* Initialisation du port SPI */
100   pinMode(10, OUTPUT); // Arduino UNO
101   //pinMode(53, OUTPUT); // Arduino Mega
102
103   /* Initialisation de la carte SD */
104   Serial.println(F("Initialisation de la carte SD ... "));
105   if (!SD.begin(SDCARD_CS_PIN)) {
106     Serial.println(F("Erreur : Impossible d'initialiser la carte SD"));
107     Serial.println(F("Verifiez la carte SD et appuyez sur le bouton
108 RESET"));
109     for (;;); // Attend appui sur bouton RESET
110   }
111
112   /* Ouvre le fichier de sortie en écriture */
```

```
113   Serial.println(F("Ouverture du fichier de sortie ... "));
114   file = SD.open(OUTPUT_FILENAME, FILE_WRITE);
115   if (!file) {
116     Serial.println(F("Erreur : Impossible d'ouvrir le fichier de
117 sortie"));
118     Serial.println(F("Verifiez la carte SD et appuyez sur le bouton
119 RESET"));
120     for (;;); // Attend appui sur bouton RESET
121   }
122
123   /* Ajoute l'entête CSV si le fichier est vide */
124   if (file.size() == 0) {
125     Serial.println(F("Ecriture de l'entete CSV ..."));
126     file.println(F("Horodatage; Tension A0; Tension A1; Tension A2;
127 Tension A3"));
128     file.flush();
129   }
130 }
131
132
133 /** Fonction loop() */
134 void loop() {
135   // Temps de la précédente mesure et actuel
136   static unsigned long previousMillis = 0;
137   unsigned long currentMillis = millis();
138
139   /* Réalise une prise de mesure toutes les DELAY_BETWEEN_MEASURES
140 millisecondes */
141   if (currentMillis - previousMillis >= DELAY_BETWEEN_MEASURES) {
142     previousMillis = currentMillis;
143     measure();
144   }
145 }
146
147
148 /** Fonction de mesure - à personnaliser selon ses besoins */
149 void measure() {
150
151   /* Lit la date et heure courante */
152   DateTime_t now;
153   read_current_datetime(&now);
154
155   /* Réalise la mesure */
156   float v_1 = analogRead(A0) * 5.0 / 1023;
157   float v_2 = analogRead(A1) * 5.0 / 1023;
158   float v_3 = analogRead(A2) * 5.0 / 1023;
159   float v_4 = analogRead(A3) * 5.0 / 1023;
160
161   /* Affiche les données sur le port série pour debug */
```

```
  Serial.print(v_1);
  Serial.print(F("; "));
  Serial.print(v_2);
  Serial.print(F("; "));
  Serial.print(v_3);
  Serial.print(F("; "));
  Serial.println(v_4);

  /* Enregistre les données sur la carte SD */
  // Horodatage
  file.print(now.days);
  file.print(F("/"));
  file.print(now.months);
  file.print(F("/"));
  file.print(now.year + 2000);
  file.print(F(" "));
  file.print(now.hours);
  file.print(F(":"));
  file.print(now.minutes);
  file.print(F(":"));
  file.print(now.seconds);
  file.print(F("; "));
  // Données brutes
  file.print(v_1);
  file.print(F("; "));
  file.print(v_2);
  file.print(F("; "));
  file.print(v_3);
  file.print(F("; "));
  file.println(v_4);
  file.flush();
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur cette page (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Articles en relation avec celui-ci

- Lire et écrire des données sur une carte SD avec une carte Arduino / Genuino
- Utiliser un module horloge temps réel DS1307 avec une carte Arduino / Genuino
- Datalogger de température autonome avec une carte Arduino et un capteur LM35