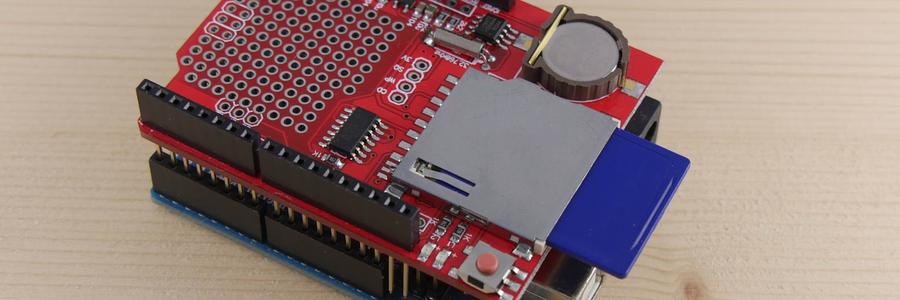
Fabriquer un système d'acquisition de mesures (datalogger) avec une carte Arduino / Genuino

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](https://www.carnetdumaker.net/membres/skywodd/) | fév. 15, 2017 | Licence (voir pied de page)

Catégories : [Tutoriels](https://www.carnetdumaker.net/articles/categories/tutoriels/) [Arduino](https://www.carnetdumaker.net/articles/categories/tutoriels/arduino/) | Mots clefs : [Arduino](https://www.carnetdumaker.net/articles/tags/arduino/) [Genuino](https://www.carnetdumaker.net/articles/tags/genuino/) [Datalogger](https://www.carnetdumaker.net/articles/tags/datalogger/) [Mesure](https://www.carnetdumaker.net/articles/tags/mesure/) [DS1307](https://www.carnetdumaker.net/articles/tags/ds1307/) [RTC](https://www.carnetdumaker.net/articles/tags/rtc/) [Carte SD](https://www.carnetdumaker.net/articles/tags/carte-sd/) [SD card](https://www.carnetdumaker.net/articles/tags/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](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/#le-principe-dun-datalogger)
* [Le montage](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/#le-montage)
* [Le code de base](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/#le-code-de-base)
* [Le code avec horodatage](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/#le-code-avec-horodatage)
* [Conclusion](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/#conclusion)

Bonjour à toutes et à tous !

Dans [un précédent article](https://www.carnetdumaker.net/articles/lire-et-ecrire-des-donnees-sur-une-carte-sd-avec-une-carte-arduino-genuino/), 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.

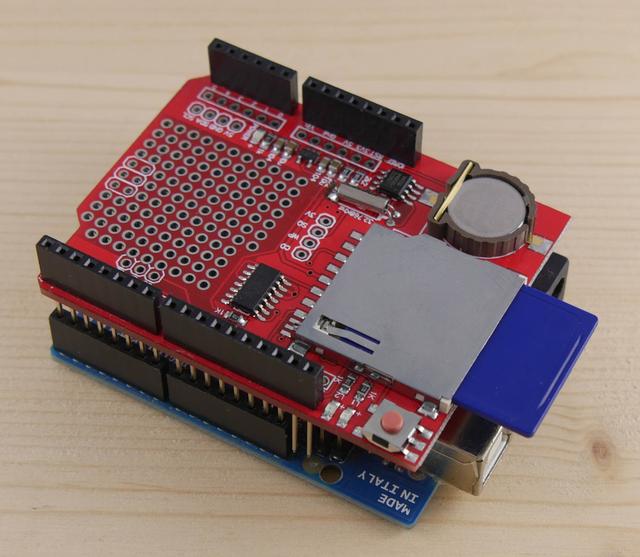
### Le principe d'un datalogger

[Un datalogger](https://fr.wikipedia.org/wiki/Enregistreur_de_donn%C3%A9es) ("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

[](https://www.carnetdumaker.net/images/datalogger-basique/)

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*](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/) et [*ici*](https://www.carnetdumaker.net/articles/lire-et-ecrire-des-donnees-sur-une-carte-sd-avec-une-carte-arduino-genuino/) 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  2  3 | */\* Dépendances \*/*  #include *<SPI.h> // Pour la communication SPI*  #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](https://www.arduino.cc/en/Reference/SPI) et [SD](https://www.arduino.cc/en/Reference/SD). Ces deux bibliothèques permettent ensemble de communiquer avec la carte SD.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | */\*\* Broche CS de la carte SD \*/*  **const** byte SDCARD\_CS\_PIN = 10; *// A remplacer suivant votre shield SD*  */\*\* Nom du fichier de sortie \*/*  **const** char\* OUTPUT\_FILENAME = "data.csv";  */\*\* Delai entre deux prises de mesures \*/*  **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  2 | */\*\* Fichier de sortie avec les mesures \*/*  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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | */\*\* 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](https://fr.wikipedia.org/wiki/Comma-separated_values). 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).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | */\*\* 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](https://www.carnetdumaker.net/articles/faire-plusieurs-choses-la-fois-avec-une-carte-arduino/) 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | */\*\* 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();  } |

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  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101 | */\*\**  *\* Exemple de code Arduino pour un datalogger basique avec stockage sur carte SD.*  *\*/*  */\* Dépendances \*/*  #include *<SPI.h> // Pour la communication SPI*  #include *<SD.h> // Pour la communication avec la carte SD*  */\*\* Broche CS de la carte SD \*/*  **const** byte SDCARD\_CS\_PIN = 10; *// A remplacer suivant votre shield SD*  */\*\* Nom du fichier de sortie \*/*  **const** char\* OUTPUT\_FILENAME = "data.csv";  */\*\* Delai entre deux prise de mesures \*/*  **const** unsigned long DELAY\_BETWEEN\_MEASURES = 5000;  */\*\* Fichier de sortie avec les mesures \*/*  File file;  */\*\* 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();  }  }  */\*\* 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();  }  }  */\*\* 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*](https://www.carnetdumaker.net/snippets/112/) (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*](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/) que je vous recommande *;)*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | */\* 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](https://www.arduino.cc/en/Reference/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.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | */\*\* 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | */\*\* 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | */\*\* 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 :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182 | */\*\**  *\* 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  #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  */\*\* Broche CS de la carte SD \*/*  **const** byte SDCARD\_CS\_PIN = 10; *// A remplacer suivant votre shield SD*  */\*\* Nom du fichier de sortie \*/*  **const** char\* OUTPUT\_FILENAME = "data.csv";  */\*\* Delai entre deux prises de mesures \*/*  **const** unsigned long DELAY\_BETWEEN\_MEASURES = 5000;  */\*\* 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;  }  */\*\* Fichier de sortie avec les mesures \*/*  File file;  */\*\* 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*  }  */\* 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("Horodatage; Tension A0; Tension A1; Tension A2; Tension A3"));  file.flush();  }  }  */\*\* 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();  }  }  */\*\* 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();  } |

L'extrait de code ci-dessus est disponible en téléchargement sur [*cette page*](https://www.carnetdumaker.net/snippets/dossiers/5/) (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](https://www.carnetdumaker.net/articles/lire-et-ecrire-des-donnees-sur-une-carte-sd-avec-une-carte-arduino-genuino/)
* [Utiliser un module horloge temps réel DS1307 avec une carte Arduino / Genuino](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/)
* [Datalogger de température autonome avec une carte Arduino et un capteur LM35](https://www.carnetdumaker.net/articles/datalogger-de-temperature-autonome-avec-une-carte-arduino-et-un-capteur-lm35/)