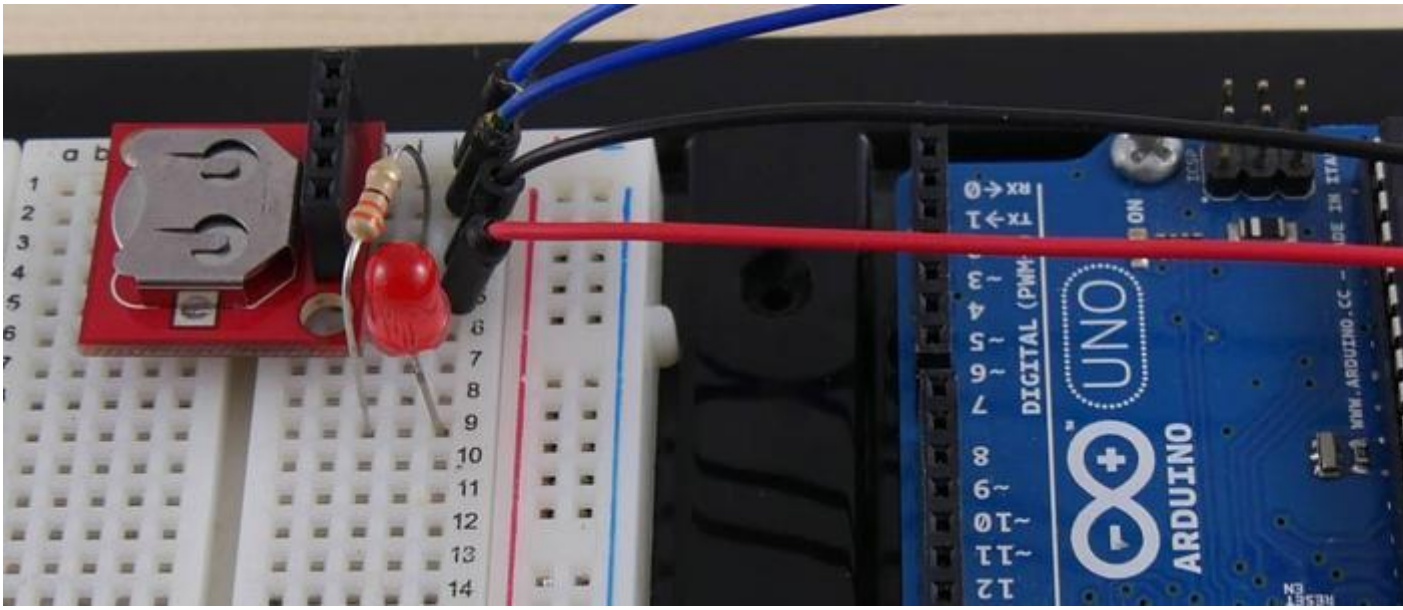
Utiliser un module horloge temps réel DS1307 avec une carte Arduino / Genuino

Avant l'heure, c'est pas l'heure, après l'heure, c'est plus l'heure

https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/



par skywodd | déc. 7, 2016 | Licence (voir pied de page)

Catégories : Tutoriels Arduino | Mots clefs : Arduino Genuino Temps Wire I2C TWI DS1307 RTC

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 apprendre ensemble à utiliser un module horloge temps réel DS1307 avec une carte Arduino / Genuino. En bonus, nous verrons comment utiliser la sortie programmable du module pour générer une base de temps. Nous

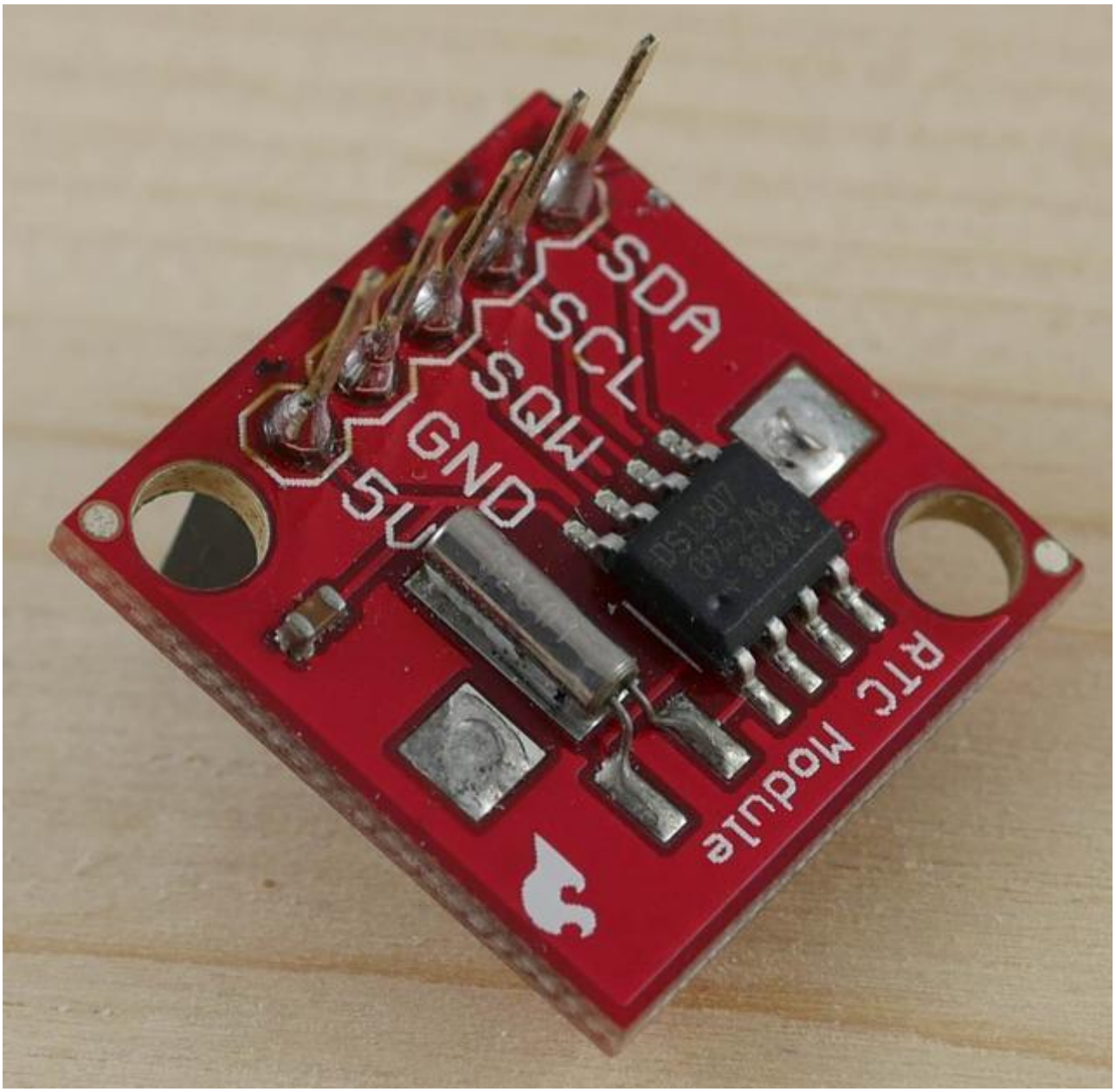
verrons aussi comment utiliser la mémoire NVRAM du module pour conserver des données.

Sommaire

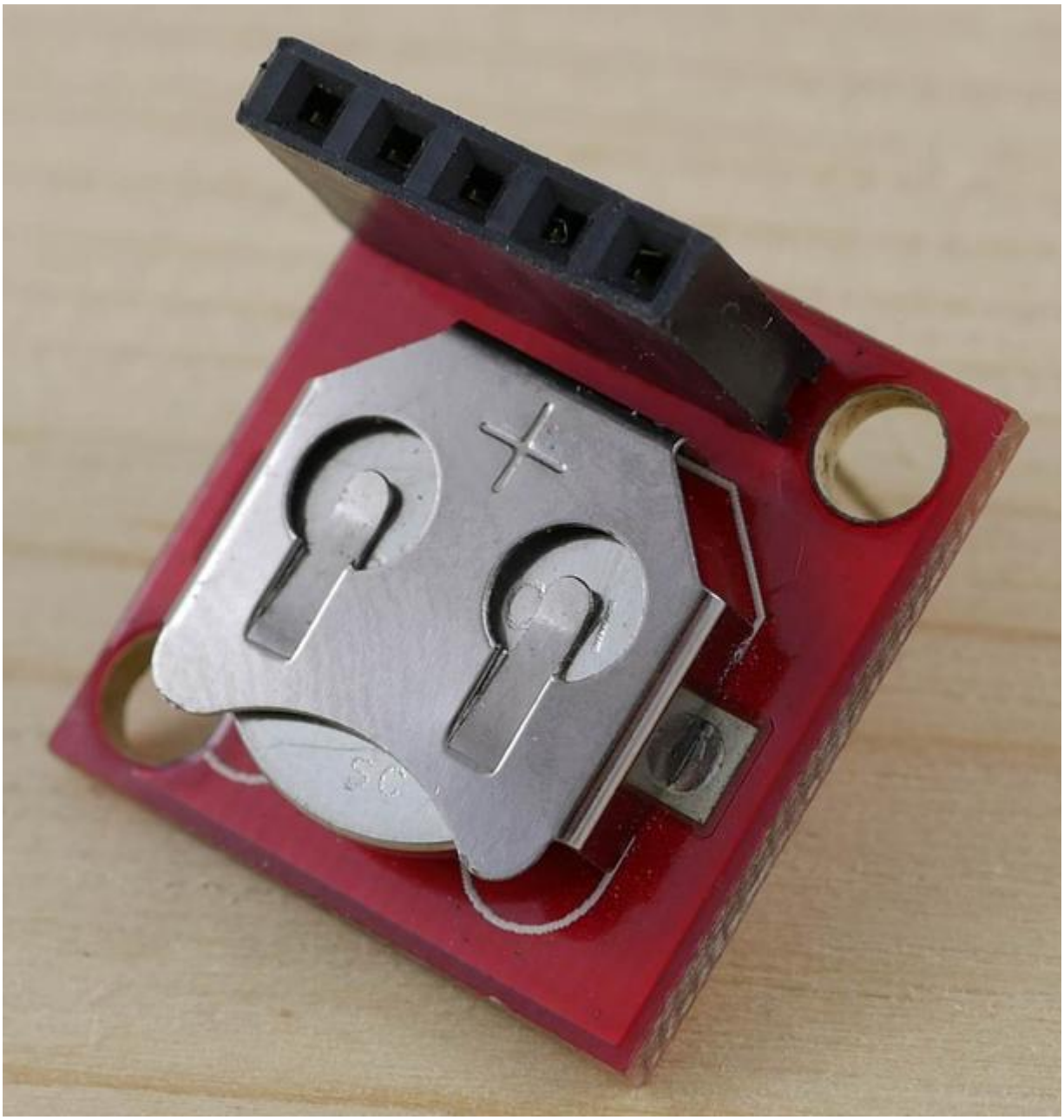
Bonjour à toutes et à tous !

Dans un monde qui fonctionne à pleine vitesse, on a besoin de connaitre l'heure. Dans ce tutoriel, on donc va prendre le temps de mesurer le temps avec une horloge temps réel.

Le module DS1307



Module DS1307



Module DS1307

Le module DS1307 de Maxim Integrated est une horloge temps réel (aussi appelé "RTC", aka "Real Time Clock"). C'est une horloge numérique autonome qui donne l'heure quand on la lui demande. Ce genre d'horloge est très utile dans des projets de mesure de grandeurs physiques avec horodatage par exemple.

N.B. Le module RTC DS1307 fonctionne en 5 volts uniquement. Pour des applications 3.3 volts, il faudra trouver une autre solution.

Ce module RTC est capable de gérer l'heure (heures, minutes, secondes) et la date (jours, mois, année) tout en s'occupant des mois de 30 ou 31 jours, des années bissextiles, etc. Le calendrier intégré dans le module DS1307 est valable de l'an 2000 à l'an 2100, ce qui devrait être suffisant pour la plupart des projets.

PS Le module dérive de quelques secondes par jours en moyenne. Cela dépend de la température ambiante et de la qualité du quartz d'horloge.

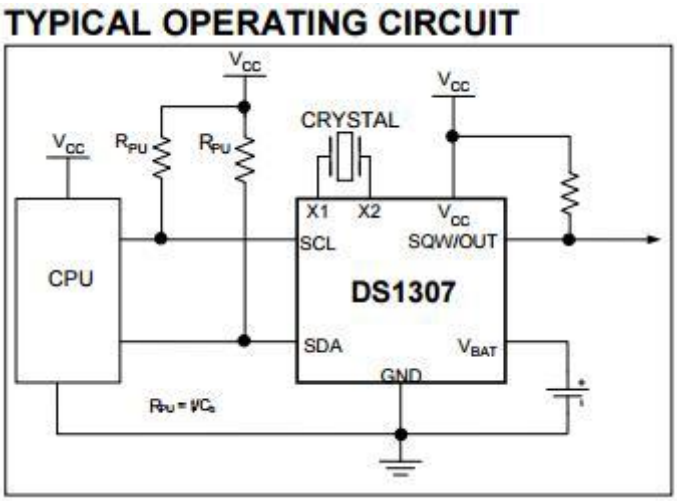


Schéma d'application typique

La communication avec le microcontrôleur maître se fait via un bus I²C. Le module dispose de tout le nécessaire pour garder en mémoire l'heure en cas de coupure d'alimentation grâce à une batterie externe. Une simple pile bouton permet de garder l'heure et la date à jour durant plusieurs années sans alimentation.

Le module DS1307 ne dispose pas de fonctionnalité "alarme" contrairement à d'autres modules RTC plus haut de gamme. Le module DS1307 dispose cependant d'une sortie "base de temps" permettant d'avoir un signal logique à une fréquence fixe (1 Hertz par exemple) pour faire fonctionner un circuit ou un compteur externe. Cela peut être utile dans certaines applications.

En bonus, le module DS1307 dispose de 56 octets de mémoire NVRAM (mémoire non volatile qui conserve son contenu tant que la batterie de secours est fonctionnelle). Il est possible d'utiliser ces quelques octets de mémoire pour stocker des données non critiques, comme l'état d'un menu par exemple.

Principe d'utilisation du module

D'un point de vue logiciel, le module DS1307 se comporte comme une petite mémoire externe I²C contenant l'heure et la date à des emplacements fixes.

La plage mémoire du module est d'une taille impressionnante de … 64 octets. Bon, OK, ce n'est pas énorme, mais il ne faut pas beaucoup de mémoire pour stocker une date et une heure. Lire la mémoire du module revient à lire l'heure et la date courante. Écrire la mémoire du module revient à mettre à jour l'heure et la date. Ce n'est pas plus compliqué que cela.

BCD : Binary Coded Decimal

Connaissez-vous l'encodage BCD, communément appelé "binary coded decimal" ou "décimal codé binaire" en bon français ?

Si oui, bonne nouvelle, si non, il va falloir apprendre le principe du BCD avant de continuer 

Si je vous demande d'écrire le nombre 42 sur une feuille de papier, vous allez me prendre pour un idiot fini et écrire deux chiffres sur une feuille : 4 et 2. Félicitation, vous savez écrire un chiffre en base 10 (décimal).

Maintenant, si je vous demande d'écrire 42 en binaire (base 2) sur cette même feuille, vous allez réfléchir un instant et écrire 0 0 1 0 1 0 1 0.

PS Si vous ne savez pas écrire un nombre décimal en binaire, allez tout de suite sur votre moteur de recherche préféré et lisez des cours sur le sujet. Savoir compter en binaire est un prérequis fondamental en informatique 

Le décimal codé binaire est un mélange bizarre entre décimal et binaire. L'idée est là suivante : on écrit un nombre en décimal puis on converti chaque chiffre en binaire (sur 4 bits) indépendamment. Exemple : 42 devient 0100 0010 en BCD.

À ce stade, vous devez sûrement vous demander qui est le crétin qui a inventé cet encodage à la noix ! Le binaire est déjà assez compliqué pour nous autres pauvres humains, pourquoi rendre les choses encore plus compliquées avec du BCD !? En plus, avec un octet on code seulement deux chiffres décimaux. Au lieu d'avoir 256 valeurs possibles, on en a que 100. C'est nul !

Pourquoi l'encodage BCD existe-t-il ? La réponse est simple : c'est plus simple à gérer électroniquement.

Prenons le cas d'une horloge vu que c'est le sujet qui nous intéresse ici. Les secondes vont de 00 à 59. Si on code les secondes en binaire, il faut 6 bits (64 valeurs possibles). Cela signifie que pour vérifier que les secondes ne dépassent pas 59, il faut tester l'état de 6 bits à la fois.

En BCD, chaque chiffre est encodé sur 4 bits. Pour vérifier que les secondes ne dépassent pas 59, il suffit de tester le chiffre des dizaines, soit 4 bits. Cela fait deux bits de moins à tester, donc moins de complexité dans la logique de contrôle.

On retrouve ce genre d'astuce un peu partout en électronique. L'encodage BCD n'est qu'une astuce parmi d'autres.

Table 2. Timekeeper Registers

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

0 = Always reads back as 0.

Registres d'un module DS1307

Les sept premiers octets de la mémoire contiennent la date et l'heure. Le huitième octet contient des données de configuration. Et les 56 octets restants sont utilisables librement par l'utilisation (voir chapitre bonus).

- Le premier octet contient le nombre de secondes courant encodé en BCD.

  Le bit de poids fort (annoté `CH` sur le schéma pour "Clock Halt" / "Arrêt Horloge") permet de connaitre l'état de l'horloge du module. Si `CH = 1`, l'horloge du module est arrêtée et par conséquent, la date et l'heure ne sont plus mises à jour. Cela se produit quand la pile du module est HS et que l'alimentation a été coupée. Dans une telle situation, il est nécessaire de mettre à jour la date et l'heure en mémoire et de remettre `CH = 0`.

- Le second octet contient le nombre de minutes courant encodé en BCD.
- Le troisième octet contient l'heure courante encodée en BCD.

  Le bit n°6 permet de choisir le format d'heure (12 heures ou 24 heures). Si le bit n°6 = 1, alors l'heure est en format 12 heures (format américain) et le bit n°5 fait office d'indicateur AM/PM (matin / après-midi). Si le bit n°6 = 0, alors le bit n°5 fait partir du codage BCD pour le chiffre des dizaines de l'heure courante.

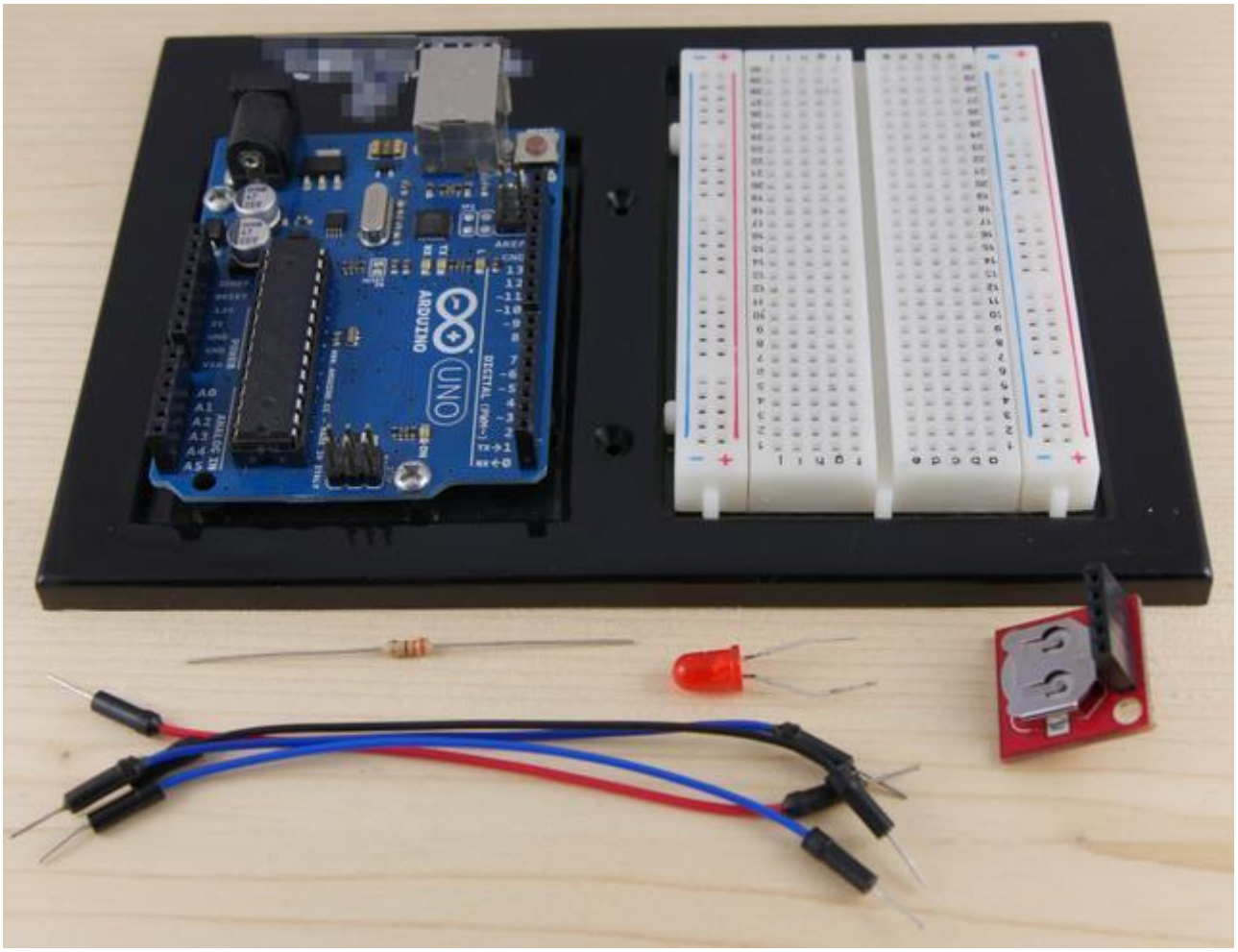
- Le quatrième octet contient le jour de la semaine courant encodé en BCD (1 pour lundi, 2 pour mardi, 3 pour mercredi, etc).
- Le cinquième octet contient le nombre de jours courant encodé en BCD.
- Le sixième octet contient le mois courant encodé en BCD.
- Le septième octet contient l'année courante encodée en BCD.

  N.B. L'année est codée sur deux chiffres. Exemple : 2016 = 16.

- Le huitième octet est un registre de configuration. Celui-ci permet de configurer le mode de fonctionnement de la broche `SQW` (voir chapitre bonus).
- Les 56 octets restants sont utilisables librement par l'utilisateur (voir chapitre bonus).

Le montage de démonstration

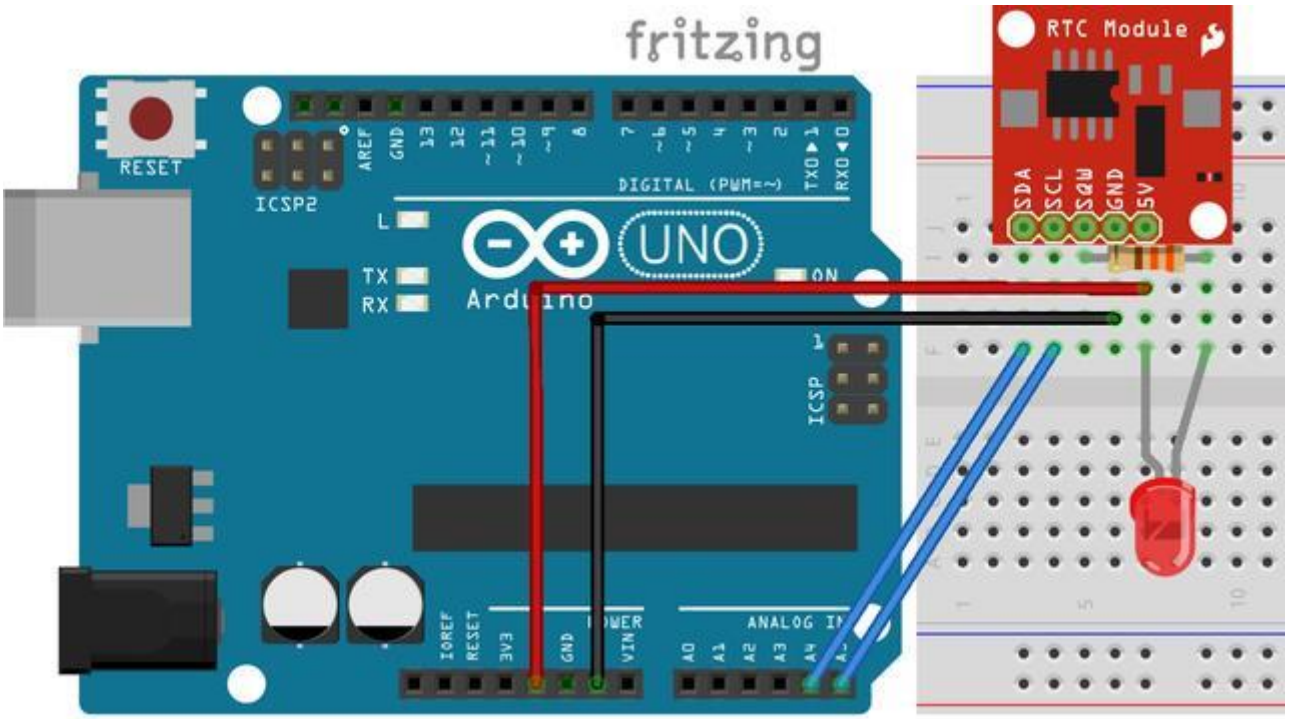
Le montage de démonstration pour ce tutoriel pourrait se résumer à empiler une shield "RTC" par-dessus une carte Arduino / Genuino. Pour rendre cet article un peu plus complet, j'ai décidé de partir sur une solution à base d'un module DS1307 "breakout" que l'on trouve sur divers sites de vente pour quelques euros.



Matériel nécessaire

Pour réaliser ce montage, il va nous falloir :

- Une carte Arduino UNO (et son câble USB),
- Un module RTC DS1307 (avec sa pile de sauvegarde),
- Une résistance de 330 ohms et une LED (pour le chapitre bonus),
- Une plaque d'essai et des fils pour câbler notre montage.



Vue prototypage du montage

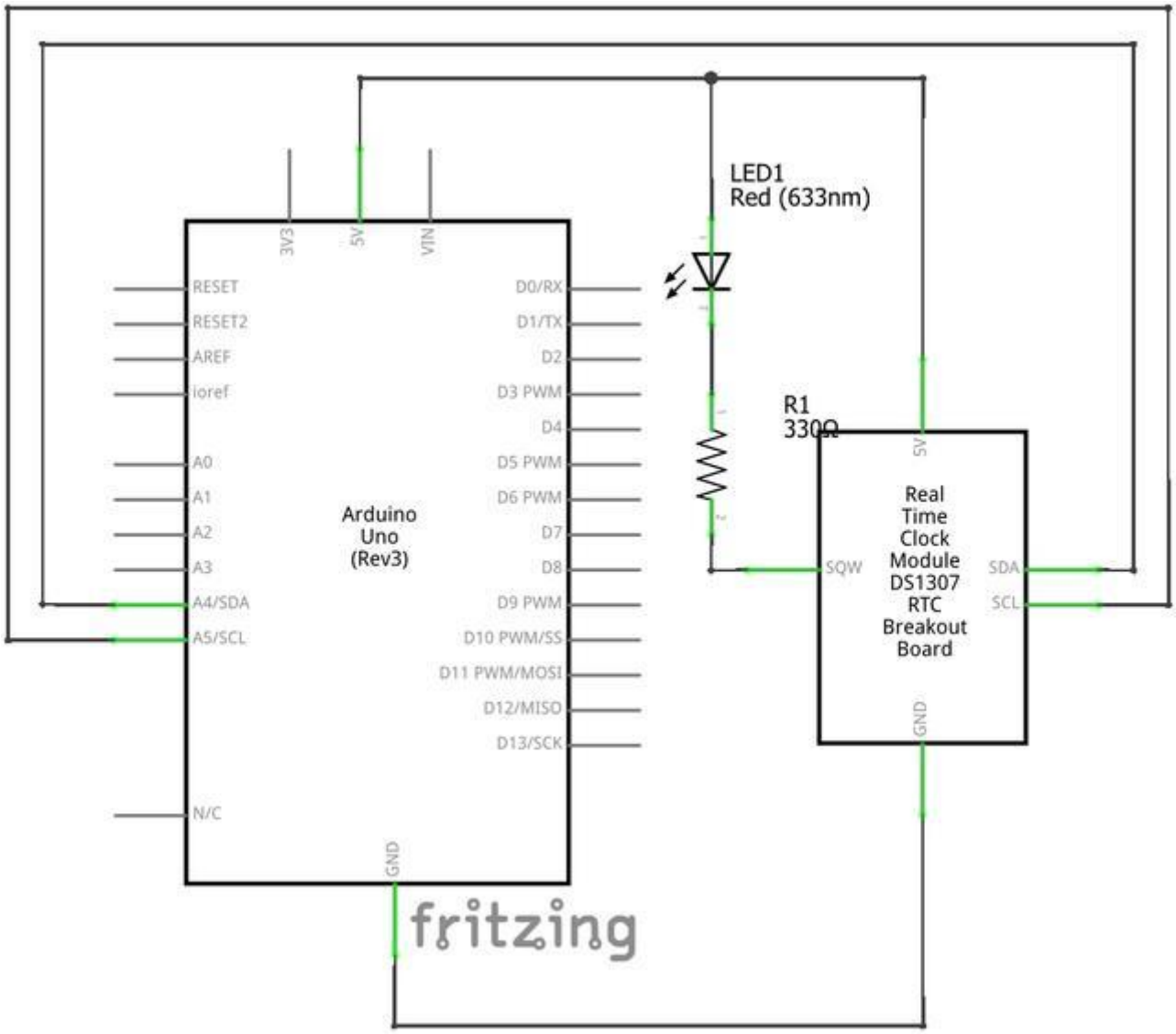
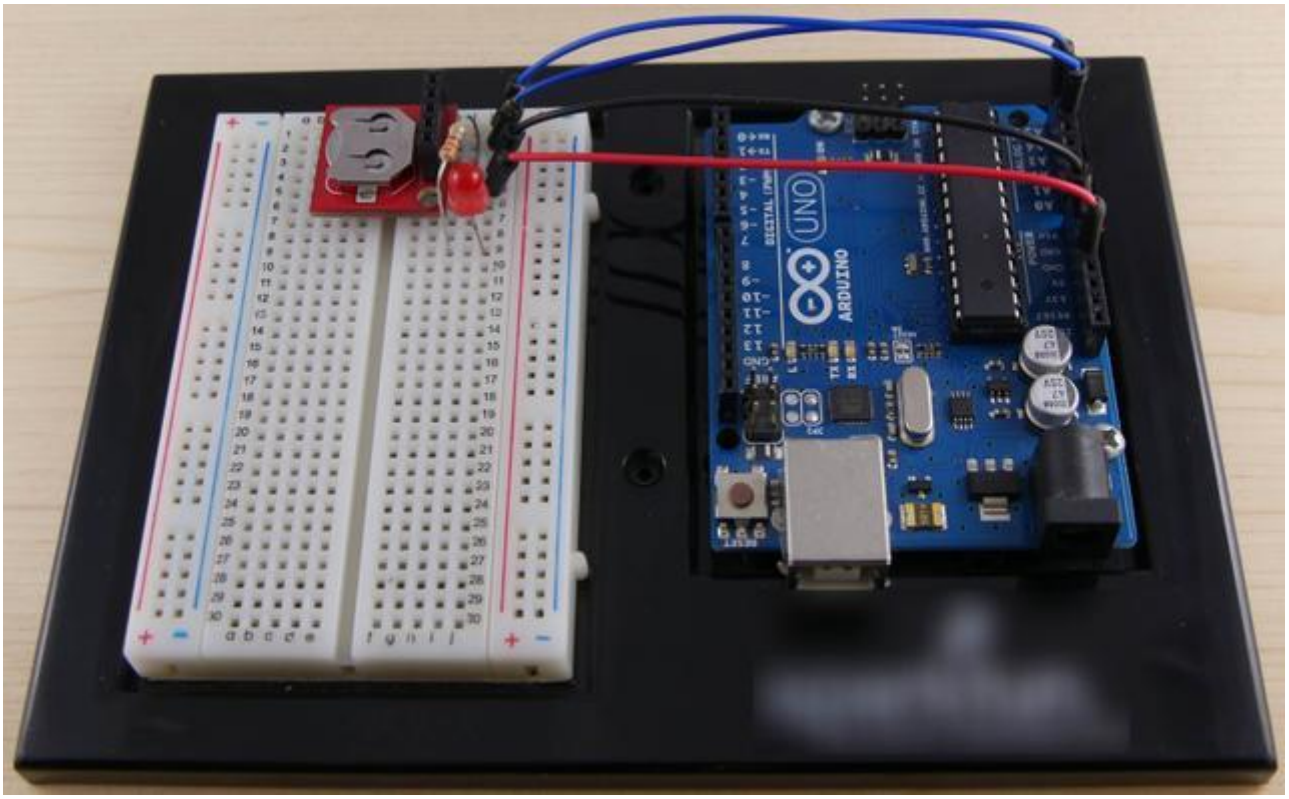


Illustration du câblage

Pour câbler le module DS1307 à la carte Arduino, il suffit de quatre fils :

- un fil entre la broche `5V` de la carte Arduino et la broche `5V/VCC` du module,
- un fil entre la broche `GND` de la carte Arduino et la broche `GND` du module,
- un fil entre la broche `A4` de la carte Arduino et la broche `SDA` du module,
- un fil entre la broche `A5` de la carte Arduino et la broche `SCL` du module.



Le montage fini

Pour ceux qui voudraient essayer le code du chapitre bonus. Il est nécessaire de câbler une LED et une résistance de 330 ohms entre la broche `SQW` du module DS1307 et la broche `5V` de la carte Arduino. La cathode (méplat) de la LED devra pointer du côté de la broche `SQW`.

N.B. Si vous ne comptez pas utiliser la broche `SQW` et le code du chapitre bonus (ce qui sera le cas dans 99.9% des cas), la LED et la résistance sont inutiles. Laissez la broche `SQW` non connectée et tout ira bien.

Le code de lecture et ajustement du temps

Avant de commencer, il est important que vous compreniez bien que les codes ci-dessous et en chapitres bonus sont en deux parties.

Chaque "projet" Arduino comporte donc deux fichiers : le fichier de code Arduino et le fichier `DS1307.h`, dans un même dossier portant le nom du projet Arduino.

Pour résumer visuellement, voici à quoi devrait ressembler un projet Arduino nommé `mon_code_arduino` :

- `mon_code_arduino` (dossier)
  - `mon_code_arduino.ino`
  - `DS1307.h`

Si le fichier `DS1307.h` ne se trouve pas au même niveau que le fichier de code Arduino, vous obtiendrez une erreur à la compilation.

Nous allons commencer le code de ce tutoriel avec la déclaration de toute une série de constantes et de structures de données. Cela va nécessiter la création d'un fichier séparé que l'on nommera `DS1307.h`. Ce fichier devra se trouver au même niveau que le fichier de code Arduino.

PS Le fichier `DS1307.h` va contenir la totalité des constantes et structures de données nécessaires pour ce chapitre et les chapitres bonus. Ne soyez donc pas choqué de voir des constantes inutilisées pour le moment 

N.B. Il est obligatoire de faire un fichier séparé pour ces constantes et structures de données, car le logiciel Arduino ne gère pas les déclarations de structures de données dans les fichiers de code portant l'extension `.ino`, `.c` ou `.cpp`.

```
1 /** Adresse I2C du module RTC DS1307 */
2 const uint8_t DS1307_ADDRESS = 0x68;
```

On commence l'écriture de notre fichier `DS1307.h` par la déclaration d'une constante contenant l'adresse I²C du module DS1307. Cette adresse est non modifiable. Elle est codée en dure dans le module et elle est identique pour tous les modules DS1307.

N.B. Cela signifie qu'il est impossible d'avoir plusieurs modules DS1307 sur un même bus I²C. Cela n'est cependant pas un problème, sauf si vous êtes du genre à avoir plusieurs montres à votre poignet.

```
1 /** Adresse du registre de contrôle du module RTC DS1307 */
2 const uint8_t DS1307_CTRL_REGISTER = 0x07;
```

On continu ensuite avec la déclaration de l'adresse du registre de configuration du module DS1307.

Il est toujours préférable d'avoir une constante plutôt qu'une valeur en dure dans le code. C'est plus lisible par la suite quand on relit son code 

```
1 /** Adresse et taille de la NVRAM du module RTC DS1307 */
2 const uint8_t DS1307_NVRAM_BASE = 0x08;
3 const uint8_t DS1307_NVRAM_SIZE = 56;
```

On ajoute ensuite deux constantes pour la gestion de la NVRAM du module que l'on verra en chapitre bonus.

La première constante définit l'adresse de début de la NVRAM et la seconde constante définit la taille de la NVRAM.

```
1 /** Structure contenant les informations de date et heure en provenance
2 ou à destination du module RTC */
3 typedef struct {
4   uint8_t seconds; /**!< Secondes 00 - 59 */
```

```
  uint8_t minutes; /**!< Minutes 00 - 59 */
  uint8_t hours;  /**!< Heures 00 - 23 (format 24h), 01 - 12 (format 12h)
*/
  uint8_t is_pm; /**!< Vaut 1 si l'heure est en format 12h et qu'il est
l'aprés midi, sinon 0 */
  uint8_t day_of_week;  /**!< Jour de la semaine 01 - 07, 1 = lundi, 2 =
mardi, etc.  */
  uint8_t days; /**!< Jours 01 - 31 */
  uint8_t months;  /**!< Mois 01 - 12 */
  uint8_t year;  /**!< Année au format yy (exemple : 16 = 2016) */
} DateTime_t;
```

Attention, ça se complique !

On déclare une structure de données qui a pour but de contenir les informations de date et heure en provenance ou à destination du module.

J'ai nommé cette structure `DateTime_t`, car elle permet de stocker une date et une heure. `_t` est une convention en programmation C pour dire "c'est un type de données".

Grâce à cette structure, il est possible de passer une date et une heure en paramètre d'une fonction d'un seul bloc, au lieu de devoir passer chaque composant de la date et de l'heure un par un.

```
/** Mode de fonctionnement pour la broche SQW */
typedef enum {
  SQW_1_HZ = 0, /**!< Signal à 1Hz sur la broche SQW */
  SQW_4096_HZ,  /**!< Signal à 4096Hz sur la broche SQW */
  SQW_8192_HZ,  /**!< Signal à 8192Hz sur la broche SQW */
  SQW_32768_HZ, /**!< Signal à 32768Hz sur la broche SQW */
  SQW_DC /**!< Broche SQW toujours à LOW ou HIGH */
} DS1307_Mode_t;
```

Pour finir, on déclare une énumération de constantes pour les différents modes de fonctionnement de la broche `SQW` que l'on verra en chapitre bonus.

Voici le code complet du fichier `DS1307.h` avec commentaires et l'include guard qui va bien (sur le Carnet du Maker ont fait les choses bien jusqu'au bout ;)) :

```
#ifndef DS1307_H
#define DS1307_H

/** Adresse I2C du module RTC DS1307 */
const uint8_t DS1307_ADDRESS = 0x68;

/** Adresse du registre de contrôle du module RTC DS1307 */
const uint8_t DS1307_CTRL_REGISTER = 0x07;

```

```
/** Adresse et taille de la NVRAM du module RTC DS1307 */
const uint8_t DS1307_NVRAM_BASE = 0x08;
const uint8_t DS1307_NVRAM_SIZE = 56;


/** Structure contenant les informations de date et heure en provenance
ou à destination du module RTC */
typedef struct {
  uint8_t seconds; /**!< Secondes 00 - 59 */
  uint8_t minutes; /**!< Minutes 00 - 59 */
  uint8_t hours;  /**!< Heures 00 - 23 (format 24h), 01 - 12 (format 12h)
*/
  uint8_t is_pm; /**!< Vaut 1 si l'heure est en format 12h et qu'il est
l'aprés midi, sinon 0 */
  uint8_t day_of_week;  /**!< Jour de la semaine 01 - 07, 1 = lundi, 2 =
mardi, etc.  */
  uint8_t days; /**!< Jours 01 - 31 */
  uint8_t months;  /**!< Mois 01 - 12 */
  uint8_t year;  /**!< Année au format yy (exemple : 16 = 2016) */
} DateTime_t;


/** Mode de fonctionnement pour la broche SQW */
typedef enum {
  SQW_1_HZ = 0, /**!< Signal à 1Hz sur la broche SQW */
  SQW_4096_HZ,  /**!< Signal à 4096Hz sur la broche SQW */
  SQW_8192_HZ,  /**!< Signal à 8192Hz sur la broche SQW */
  SQW_32768_HZ, /**!< Signal à 32768Hz sur la broche SQW */
  SQW_DC /**!< Broche SQW toujours à LOW ou HIGH */
} DS1307_Mode_t;

#endif /* DS1307_H */
```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#).

Vous êtes toujours vivant ? Oui ? Super.

On va pouvoir maintenant voir comment lire et modifier la date et l'heure dans le module.

```
/* Dépendances */
#include <Wire.h>
#include "DS1307.h"
```

On va commencer notre code Arduino avec l'inclusion des deux dépendances de notre projet : le fichier `DS1307.h` que l'on vient de créer et [la bibliothèque Wire](#) fournie de base avec le logiciel Arduino qui va nous permettre de communiquer en I²C avec le module DS1307.

N.B. Vous remarquerez que l'on utilise la syntaxe `#include <XXX.h>` pour inclure une bibliothèque de code et `#include "XXX.h"` pour inclure un fichier local au projet.

```
/* 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
```

On va ensuite devoir ajouter un peu de magie noire pour gérer indifféremment les versions 00XX et 1.X du logiciel Arduino.

Je ne pense pas que beaucoup d'entre vous utilise encore la version 0023 du logiciel Arduino, mais dans le doute, je préfère garder la rétrocompatibilité avec les anciennes versions.

Ce que fait le petit extrait de code ci-dessus, c'est remplacer les fonctions `Wire_write()` et `Wire_read()` dans le code par les véritables fonctions Arduino correspondantes en fonction de la version du logiciel Arduino utilisée.

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

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

On va devoir ensuite déclarer deux fonctions permettant de convertir un nombre décimal en BCD et inversement. Les fonctions de lecture / écriture de la date et heure pourront ainsi faire la conversion sans que notre code utilisateur ait à gérer le moindre nombre en BCD.

```
/**
 * 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;
}
```

La fonction `read_current_datetime()` prend en argument un pointeur vers une structure `DateTime_t` que l'on a déclarée précédemment dans le fichier `DS1307.h`. Cette fonction lit la mémoire du module DS1307, extrait les informations de date et heure et copie ses informations (en décimal) dans la structure.

En bonus, cette fonction retourne une valeur : zéro si l'horloge fonctionne, différent de zéro si l'horloge est arrêtée. Il est ainsi possible de détecter un problème de pile / alimentation lors de la lecture de la date / heure.

La fonction débute une transaction I²C au moyen de la fonction Wire.beginTransmission(), puis écrit un octet correspondant à l'adresse mémoire à partir de laquelle on souhaite lire la mémoire et conclut la transaction en appelant Wire.endTransmission().

La lecture de la mémoire se fait au moment de l'appel à Wire.requestFrom() qui demande au module DS1307 de lui retourner 7 octets de données à partir de l'adresse mémoire précédemment sélectionnée.

Le reste du code n'est que du décodage BCD et de la logique pour gérer le format 12 heures et 24 heures correctement.

```
/**
 * Fonction ajustant l'heure et la date courante du module RTC à partir
des informations fournies.
 * N.B. Redémarre l'horloge du module RTC si nécessaire.
```

```
 */
void adjust_current_datetime(DateTime_t *datetime) {

  /* Début de la transaction I2C */
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire.write((byte) 0); // Ecriture mémoire à l'adresse 0x00
  Wire.write(decimal_to_bcd(datetime->seconds) & 127); // CH = 0
  Wire.write(decimal_to_bcd(datetime->minutes));
  Wire.write(decimal_to_bcd(datetime->hours) & 63); // Mode 24h
  Wire.write(decimal_to_bcd(datetime->day_of_week));
  Wire.write(decimal_to_bcd(datetime->days));
  Wire.write(decimal_to_bcd(datetime->months));
  Wire.write(decimal_to_bcd(datetime->year));
  Wire.endTransmission(); // Fin de transaction I2C
}
```

La fonction `adjust_current_datetime()` prend en argument un pointeur vers une structure `DateTime_t` comme avec la fonction `read_current_datetime()`. Cette fonction écrit la mémoire du module DS1307 à partir des informations de date et heure fournies dans la structure.

La fonction débute une transaction I²C au moyen de la fonction Wire.beginTransmission(), puis écrit un octet correspondant à l'adresse mémoire à partir de laquelle on souhaite écrire la mémoire, suivi des 7 octets de données de l'heure et de la date (réencodé en BCD) et conclut la transaction en appelant Wire.endTransmission().

N.B. La fonction `adjust_current_datetime()` prend soin de remettre le bit `CH` à zéro pour que l'horloge du module redémarre si cela est nécessaire.

PS La fonction part du principe que l'heure est au format 24 heures.

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

  /* Initialise le port série */
  Serial.begin(115200);

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

  /* Vérifie si le module RTC est initialisé */
  DateTime_t now;
  if (read_current_datetime(&now)) {
    Serial.println(F("L'horloge du module RTC n'est pas active !"));

    // Reconfiguration avec une date et heure en dure (pour l'exemple)
    now.seconds = 0;
    now.minutes = 0;
    now.hours = 12; // 12h 0min 0sec
```

```
    now.is_pm = 0;
    now.day_of_week = 4;
    now.days = 1;
    now.months = 12;
    now.year = 16; // 1 dec 2016
    adjust_current_datetime(&now);
  }
}
```

La fonction `setup()` initialise le port série et le bus I²C. Elle vérifie aussi si le module DS1307 est initialisé en faisant une première lecture de la date et de l'heure. Si le module n'est pas initialisé, le code utilise une date et une heure codée en dure dans le code pour reconfigurer l'horloge.

PS On verra dans un prochain article comment permettre le réglage de l'heure par l'utilisateur directement, sans avoir à coder en dure des valeurs.

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

  /* Lit la date et heure courante */
  DateTime_t now;
  if (read_current_datetime(&now)) {
    Serial.println(F("L'horloge du module RTC n'est pas active !"));
  }

  /* Affiche la date et heure courante */
  Serial.print(F("Date : "));
  Serial.print(now.days);
  Serial.print(F("/"));
  Serial.print(now.months);
  Serial.print(F("/"));
  Serial.print(now.year + 2000);
  Serial.print(F("  Heure : "));
  Serial.print(now.hours);
  Serial.print(F(":"));
  Serial.print(now.minutes);
  Serial.print(F(":"));
  Serial.println(now.seconds);

  /* Rafraichissement une fois par seconde */
  delay(1000);
}
```

La fonction `loop()` se contente de lire la date et heure courante chaque seconde et d'afficher le tout sur le port série.

Le code complet avec commentaires :

```
/**
 * Exemple de code de lecture et d'ajustement de l'heure avec un module
RTC DS1307.
 * Compatible Arduino 0023 et Arduino 1.x (et supérieur).
 */

/* Dépendances */
#include <Wire.h>
#include "DS1307.h"


/* 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


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

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


/**
 * 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;
}


/**
 * Fonction ajustant l'heure et la date courante du module RTC à partir
des informations fournies.
 * N.B. Redémarre l'horloge du module RTC si nécessaire.
 */
void adjust_current_datetime(DateTime_t *datetime) {

  /* Début de la transaction I2C */
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire_write((byte) 0); // Ecriture mémoire à l'adresse 0x00
  Wire_write(decimal_to_bcd(datetime->seconds) & 127); // CH = 0
  Wire_write(decimal_to_bcd(datetime->minutes));
  Wire_write(decimal_to_bcd(datetime->hours) & 63); // Mode 24h
  Wire_write(decimal_to_bcd(datetime->day_of_week));
  Wire_write(decimal_to_bcd(datetime->days));
  Wire_write(decimal_to_bcd(datetime->months));
  Wire_write(decimal_to_bcd(datetime->year));
  Wire.endTransmission(); // Fin de transaction I2C
}


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

  /* Initialise le port série */
  Serial.begin(115200);

  /* Initialise le port I2C */
```

```
  Wire.begin();

  /* Vérifie si le module RTC est initialisé */
  DateTime_t now;
  if (read_current_datetime(&now)) {
    Serial.println(F("L'horloge du module RTC n'est pas active !"));

    // Reconfiguration avec une date et heure en dure (pour l'exemple)
    now.seconds = 0;
    now.minutes = 0;
    now.hours = 12; // 12h 0min 0sec
    now.is_pm = 0;
    now.day_of_week = 4;
    now.days = 1;
    now.months = 12;
    now.year = 16; // 1 dec 2016
    adjust_current_datetime(&now);
  }
}


/** Fonction loop() */
void loop() {

  /* Lit la date et heure courante */
  DateTime_t now;
  if (read_current_datetime(&now)) {
    Serial.println(F("L'horloge du module RTC n'est pas active !"));
  }

  /* Affiche la date et heure courante */
  Serial.print(F("Date : "));
  Serial.print(now.days);
  Serial.print(F("/"));
  Serial.print(now.months);
  Serial.print(F("/"));
  Serial.print(now.year + 2000);
  Serial.print(F("  Heure : "));
  Serial.print(now.hours);
  Serial.print(F(":"));
  Serial.print(now.minutes);
  Serial.print(F(":"));
  Serial.println(now.seconds);

  /* Rafraichissement une fois par seconde */
  delay(1000);
}
```

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).

Bonus : Utiliser la sortie programmable du module

7	SQW/OUT	Square Wave/Output Driver. When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square-wave frequencies (1Hz, 4kHz, 8kHz, 32kHz). The SQW/OUT pin is open drain and requires an external pullup resistor. SQW/OUT operates with either V_{CC} or V_{BAT} applied. The pullup voltage can be up to 5.5V regardless of the voltage on V_{CC}. If not used, this pin can be left floating.

Description de la broche SQW

Le module DS1307 dispose d'une broche nommée `SQW` sur laquelle il est possible de faire sortir un signal d'horloge. Cela est très utile quand on dispose d'un circuit externe (comme un compteur par exemple) ayant besoin d'une base de temps fiable.

Le mode de fonctionnement de la broche `SQW` se contrôle via le registre à l'adresse `0x07` du module DS1307.

CONTROL REGISTER

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
OUT	0	0	SQWE	0	0	RS1	RS0

Bit 7: Output Control (OUT). This bit controls the output level of the SQW/OUT pin when the square-wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0. On initial application of power to the device, this bit is typically set to a 0.

Bit 4: Square-Wave Enable (SQWE). This bit, when set to logic 1, enables the oscillator output. The frequency of the square-wave output depends upon the value of the RS0 and RS1 bits. With the square-wave output set to 1Hz, the clock registers update on the falling edge of the square wave. On initial application of power to the device, this bit is typically set to a 0.

Bits 1 and 0: Rate Select (RS[1:0]). These bits control the frequency of the square-wave output when the square-wave output has been enabled. The following table lists the square-wave frequencies that can be selected with the RS bits. On initial application of power to the device, these bits are typically set to a 1.

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

Description du registre de contrôle de la broche SQW

En manipulant les bits `RS0`, `RS1`, `OUT` et `SQWE` du registre, il est possible de générer sur la broche `SQW` :

- Un signal `LOW`,
- Un signal `HIGH`,
- Un signal carré à 1 Hz,

- Un signal carré à 4 096 Hz,
- Un signal carré à 8 192 Hz,
- Un signal carré à 32 768 Hz.

N.B. La broche `SQW` est une sortie à collecteur ouvert. Une résistance de tirage est nécessaire pour l'utiliser comme une sortie logique.

Voici un code d'exemple qui configure la broche `SQW` du module DS1307 en mode "signal carré à 1Hz" (à utiliser avec une LED par exemple pour observer le changement d'état) :

```
/**
 * Exemple de code de manipulation de la broche SQW d'un module RTC DS1307.
 * Compatible Arduino 0023 et Arduino 1.x (et supérieur).
 */

/* Dépendances */
#include <Wire.h>
#include "DS1307.h"


/* 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


/**
 * Fonction de sélection du mode de fonctionnement et de la polarité (HIGH ou LOW) de la broche SQW.
 */
void set_sqw_pin_mode(DS1307_Mode_t mode, byte polarity) {

  /* Calcul la valeur du registre de contrôle */
  byte ctrl_register = !!polarity << 7; // OUT = polarity
  if (mode != SQW_DC) {
    ctrl_register |= 1 << 4; // SQWE = 1
    ctrl_register |= mode & 3; // RSx = mode
  }

  /* Début de la transaction I2C */
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire_write(DS1307_CTRL_REGISTER); // Ecriture mémoire à l'adresse 0x07
  Wire_write(ctrl_register);
  Wire.endTransmission(); // Fin de transaction I2C
```

```
}


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

  /* Initialise le port série */
  Serial.begin(115200);

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

  /* Modification la configuration de la broche SQW */
  //set_sqw_pin_mode(SQW_DC, LOW);
  //set_sqw_pin_mode(SQW_DC, HIGH);
  set_sqw_pin_mode(SQW_1_HZ, 0);
  //set_sqw_pin_mode(SQW_4096_HZ, 0);
  //set_sqw_pin_mode(SQW_8192_HZ, 0);
  //set_sqw_pin_mode(SQW_32768_HZ, 0);
}


/** Fonction loop() */
void loop() {
  // Rien à faire
}
```

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).

Bonus : Utiliser la mémoire NVRAM du module

Le module DS1307 dispose de 56 octets de mémoire RAM sauvegardés par batterie (NVRAM). Ces quelques octets de mémoire sont bien pratiques pour stocker des données non critiques comme des préférences utilisateurs par exemple.

Une mémoire NVRAM n'est pas adaptée pour stocker des données de calibration ou de configuration, car en cas de double défaut d'alimentation (alimentation + batterie), les données sont définitivement perdues. Cela reste cependant une alternative pratique à une mémoire EEPROM pour certaines applications. De plus, une mémoire NVRAM peut être écrite sans limitation et sans dégradation des performances.

```
/** Fonction de lecture de la mémoire non volatile du module RTC (56
octets maximum) */
int read_nvram_memory(byte address) {

  /* Ne lit pas en dehors de la NVRAM */
  if (address > DS1307_NVRAM_SIZE)
    return -1;
```

```
 8
 9   /* Début de la transaction I2C */
10   Wire.beginTransmission(DS1307_ADDRESS);
11   Wire_write(DS1307_NVRAM_BASE + address); // Lecture mémoire NVRAM
12   Wire.endTransmission(); // Fin de la transaction I2C
13
14   /* Lit un octet depuis la mémoire du module RTC */
15   Wire.requestFrom(DS1307_ADDRESS, (byte) 1);
16   return Wire_read();
}
```

Pour lire un octet de la NVRAM du module DS1307, il suffit de demander un octet depuis l'adresse de base de la NVRAM + l'adresse souhaitée (de 0 à 56).

```
1  /** Fonction d'écriture de la mémoire non volatile du module RTC (56
2  octets maximum) */
3  void write_nvram_memory(byte address, byte data) {
4
5    /* N'écrit pas en dehors de la NVRAM */
6    if (address > DS1307_NVRAM_SIZE)
7      return;
8
9    /* Début de la transaction I2C */
10   Wire.beginTransmission(DS1307_ADDRESS);
11   Wire_write(DS1307_NVRAM_BASE + address); // Ecriture mémoire NVRAM
12   Wire_write(data);
13   Wire.endTransmission(); // Fin de transaction I2C
}
```

L'écriture se passe de la même façon que la lecture, mais dans l'autre sens.

```
1  /** Fonction setup() */
2  void setup() {
3
4    /* Initialise le port I2C */
5    Serial.begin(115200);
6
7    /* Initialise le port I2C */
8    Wire.begin();
9
10   /* Lit la valeur actuelle */
11   byte value = read_nvram_memory(0);
12   Serial.print(F("Valeur actuelle : "));
13   Serial.println(value);
14
15   /* Met à jour la valeur */
16   Serial.print(F("Nouvelle valeur : "));
17   Serial.println(value + 1);
```

```
  write_nvram_memory(0, value + 1);
}


/** Fonction loop() */
void loop() {
  // Rien à faire
}
```

Pour l'exemple, voici un code qui incrémente l'octet à l'adresse `0x00` de la NVRAM du module à chaque démarrage de la carte Arduino. Appuyez sur le bouton `RESET` de la carte Arduino pour voir la valeur évoluer.

Le code complet avec commentaires :

```
/**
 * Exemple de code de lecture et de manipulation de la NVRAM d'un module
RTC DS1307.
 * Compatible Arduino 0023 et Arduino 1.x (et supérieur).
 */

/* Dépendances */
#include <Wire.h>
#include "DS1307.h"


/* 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


/** Fonction de lecture de la mémoire non volatile du module RTC (56
octets maximum) */
int read_nvram_memory(byte address) {

  /* Ne lit pas en dehors de la NVRAM */
  if (address > DS1307_NVRAM_SIZE)
    return -1;

  /* Début de la transaction I2C */
  Wire.beginTransmission(DS1307_ADDRESS);
  Wire_write(DS1307_NVRAM_BASE + address); // Lecture mémoire NVRAM
  Wire.endTransmission(); // Fin de la transaction I2C
```

```
34
35    /* Lit un octet depuis la mémoire du module RTC */
36    Wire.requestFrom(DS1307_ADDRESS, (byte) 1);
37    return Wire_read();
38  }
39
40
41  /** Fonction d'écriture de la mémoire non volatile du module RTC (56
42  octets maximum) */
43  void write_nvram_memory(byte address, byte data) {
44
45    /* N'écrit pas en dehors de la NVRAM */
46    if (address > DS1307_NVRAM_SIZE)
47      return;
48
49    /* Début de la transaction I2C */
50    Wire.beginTransmission(DS1307_ADDRESS);
51    Wire_write(DS1307_NVRAM_BASE + address); // Ecriture mémoire NVRAM
52    Wire_write(data);
53    Wire.endTransmission(); // Fin de transaction I2C
54  }
55
56
57  /** Fonction setup() */
58  void setup() {
59
60    /* Initialise le port I2C */
61    Serial.begin(115200);
62
63    /* Initialise le port I2C */
64    Wire.begin();
65
66    /* Lit la valeur actuelle */
67    byte value = read_nvram_memory(0);
68    Serial.print(F("Valeur actuelle : "));
69    Serial.println(value);
70
71    /* Met à jour la valeur */
72    Serial.print(F("Nouvelle valeur : "));
73    Serial.println(value + 1);
74    write_nvram_memory(0, value + 1);
75  }
76
77
78  /** Fonction loop() */
   void loop() {
     // Rien à faire
   }
```

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).

Conclusion

Ce tutoriel est désormais terminé.

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

Articles pouvant vous intéresser

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