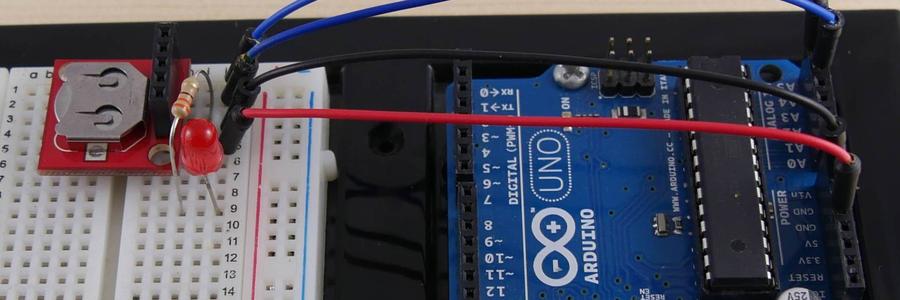
Utiliser un module horloge temps réel DS1307 avec une carte Arduino / Genuino

*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](https://www.carnetdumaker.net/membres/skywodd/) | déc. 7, 2016 | 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/) [Temps](https://www.carnetdumaker.net/articles/tags/temps/) [Wire](https://www.carnetdumaker.net/articles/tags/wire/) [I2C](https://www.carnetdumaker.net/articles/tags/i2c/) [TWI](https://www.carnetdumaker.net/articles/tags/twi/) [DS1307](https://www.carnetdumaker.net/articles/tags/ds1307/) [RTC](https://www.carnetdumaker.net/articles/tags/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.**

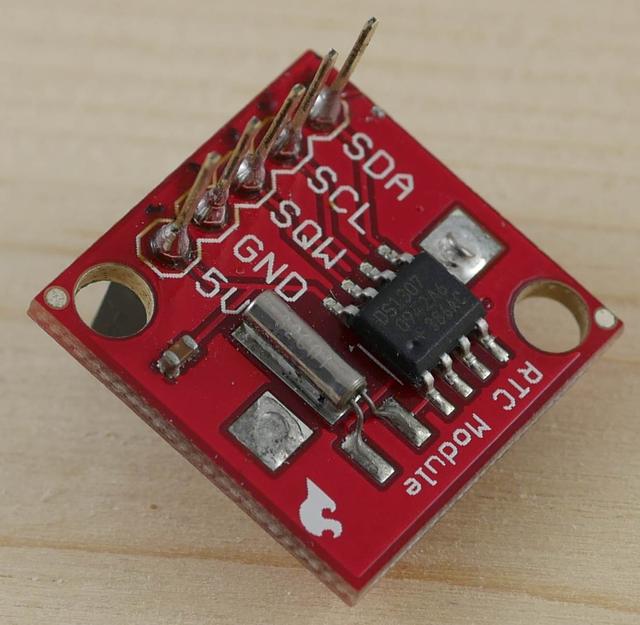
### Sommaire

* [Le module DS1307](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#le-module-ds1307)
* [Principe d'utilisation du module](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#principe-dutilisation-du-module)
* [Le montage de démonstration](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#le-montage-de-demonstration)
* [Le code de lecture et ajustement du temps](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#le-code-de-lecture-et-ajustement-du-temps)
* [Bonus : Utiliser la sortie programmable du module](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#bonus-utiliser-la-sortie-programmable-du-module)
* [Bonus : Utiliser la mémoire NVRAM du module](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#bonus-utiliser-la-memoire-nvram-du-module)
* [Conclusion](https://www.carnetdumaker.net/articles/utiliser-un-module-horloge-temps-reel-ds1307-avec-une-carte-arduino-genuino/#conclusion)

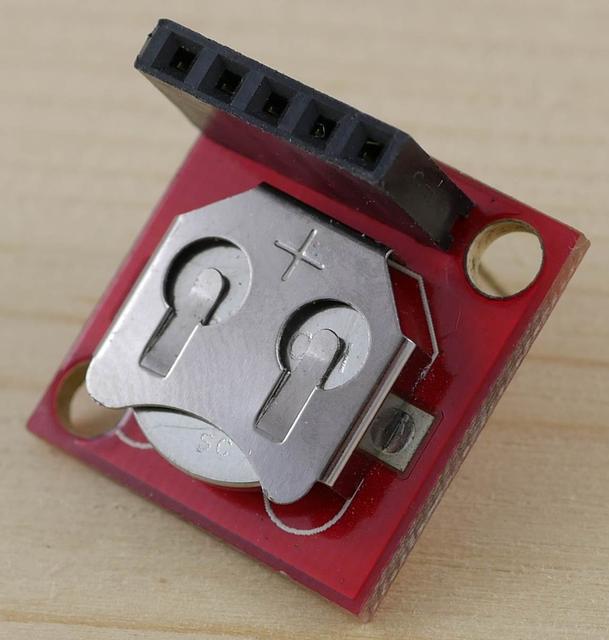
Bonjour à toutes et à tous !

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

[](https://www.carnetdumaker.net/images/module-ds1307-face-avant/)

Module DS1307

[](https://www.carnetdumaker.net/images/module-ds1307-face-arriere/)

Module DS1307

Le module [DS1307 de Maxim Integrated](https://datasheets.maximintegrated.com/en/ds/DS1307.pdf) 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.

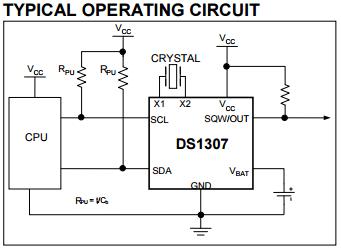
[](https://www.carnetdumaker.net/images/application-typique-dun-ds1307/)

Schéma d'application typique

La communication avec le microcontrôleur maître se fait via un [bus I²C](https://fr.wikipedia.org/wiki/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](https://fr.wikipedia.org/wiki/D%C3%A9cimal_cod%C3%A9_binaire), 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)](https://fr.wikipedia.org/wiki/Syst%C3%A8me_d%C3%A9cimal).

Maintenant, si je vous demande d'écrire 42 en [binaire (base 2)](https://fr.wikipedia.org/wiki/Syst%C3%A8me_binaire) 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.

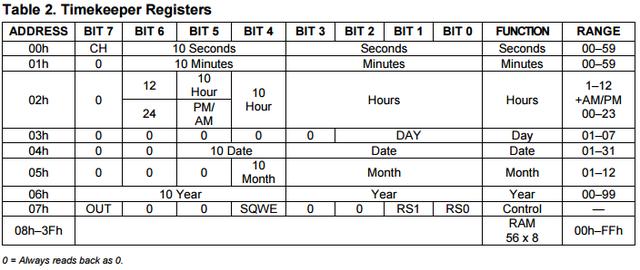
À ce stade, vous devez sûrement vous demander qui est le crétin qui a inventé cet encodage à la noix ! Le binaire est déjà assez compliqué pour nous autres pauvres humains, pourquoi rendre les choses encore plus compliquées avec du BCD !? En plus, avec un octet on code seulement deux chiffres décimaux. 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.

[](https://www.carnetdumaker.net/images/registres-module-ds1307/)

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.

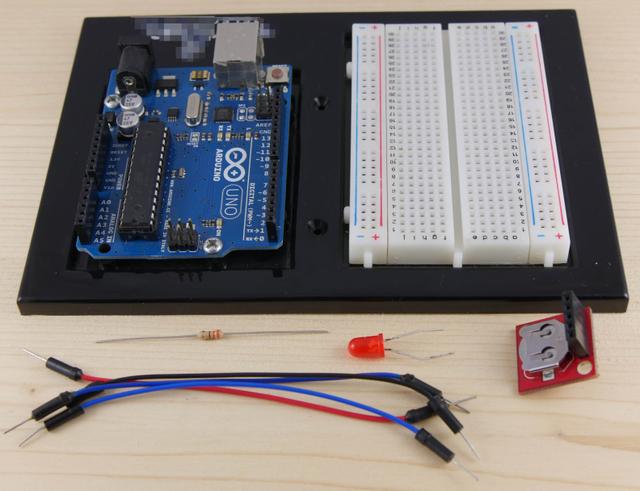
* Le quatrième octet contient le jour de la semaine courant encodé en BCD (1 pour lundi, 2 pour mardi, 3 pour mercredi, etc).
* Le cinquième octet contient le nombre de jours courant encodé en BCD.
* Le sixième octet contient le mois courant encodé en BCD.
* Le septième octet contient l'année courante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

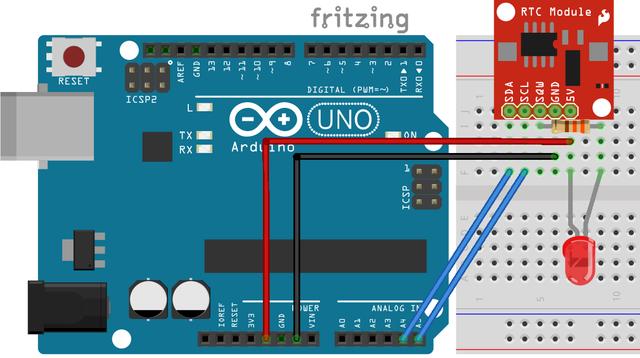
Le montage de démonstration pour ce tutoriel pourrait se résumer à empiler une shield "RTC" par-dessus une carte Arduino / Genuino. Pour rendre cet article un peu plus complet, j'ai décidé de partir sur une solution à base d'un module DS1307 "breakout" que l'on trouve sur divers sites de vente pour quelques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

[](https://www.carnetdumaker.net/images/photographie-du-materiel-necessaire-la-realisation-du-montage-de-demonstration-pour-le-module-ds1307/)

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.

[](https://www.carnetdumaker.net/images/vue-prototypage-du-montage-de-demonstration-pour-le-module-ds1307/)

Vue prototypage du montage

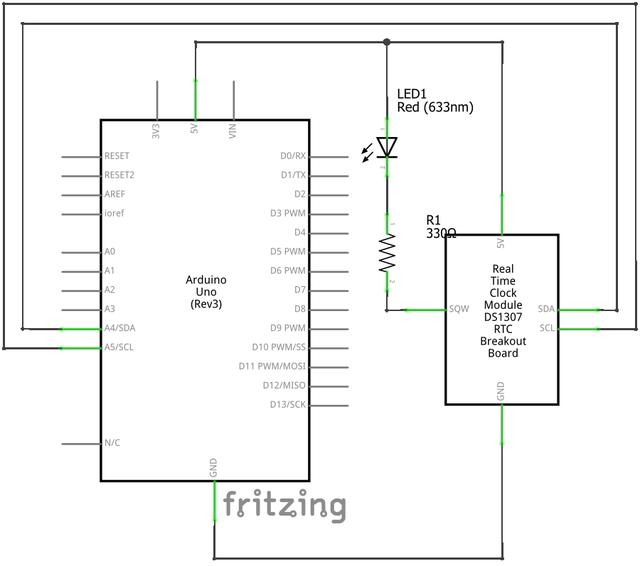
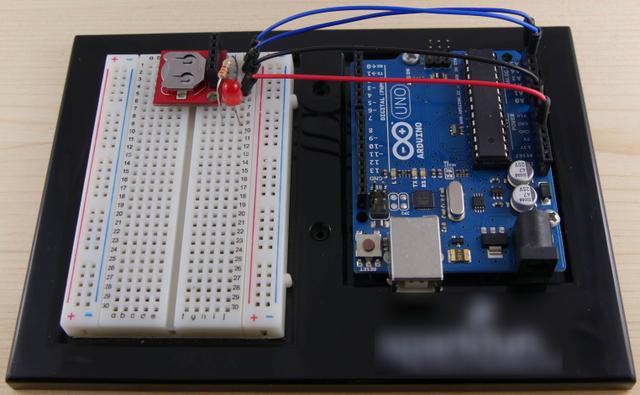
[](https://www.carnetdumaker.net/images/illustration-du-cablage-du-montage-de-demonstration-pour-le-module-ds1307/)

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.

[](https://www.carnetdumaker.net/images/photographie-du-montage-de-demonstration-pour-le-module-ds1307/)

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  2 | */\*\* Adresse I2C du module RTC DS1307 \*/*  **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  2 | */\*\* Adresse du registre de contrôle du module RTC DS1307 \*/*  **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  2  3 | */\*\* Adresse et taille de la NVRAM du module RTC DS1307 \*/*  **const** uint8\_t DS1307\_NVRAM\_BASE = 0x08;  **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  2  3  4  5  6  7  8  9  10  11 | */\*\* 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; |

Attention, ça se complique !

On déclare [une structure de données](http://www.commentcamarche.net/contents/119-les-structures-en-langage-c) 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | */\*\* 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](https://fr.wikipedia.org/wiki/Include_guard) qui va bien (sur le Carnet du Maker ont fait les choses bien jusqu'au bout ;)) :

|  |  |
| --- | --- |
| 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 | #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*](https://www.carnetdumaker.net/snippets/107/).

**Vous êtes toujours vivant ? Oui ? Super.**

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

|  |  |
| --- | --- |
| 1  2  3 | */\* 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](https://www.arduino.cc/en/Reference/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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | */\* 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](https://www.arduino.cc/en/Reference/Changes).

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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | */\*\* 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.

|  |  |
| --- | --- |
| 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 | */\*\**  *\* 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()](https://www.arduino.cc/en/Reference/WireBeginTransmission), 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()](https://www.arduino.cc/en/Reference/WireEndTransmission).

La lecture de la mémoire se fait au moment de l'appel à [Wire.requestFrom()](https://www.arduino.cc/en/Reference/WireRequestFrom) 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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | */\*\**  *\* 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()](https://www.arduino.cc/en/Reference/WireBeginTransmission), 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()](https://www.arduino.cc/en/Reference/WireEndTransmission).

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.

|  |  |
| --- | --- |
| 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 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.

|  |  |
| --- | --- |
| 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 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);  } |

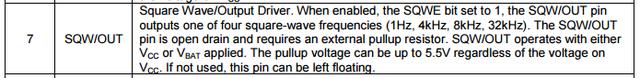
La fonction loop() se contente de lire la date et heure courante chaque seconde et d'afficher le tout sur le port série.

Le code complet avec commentaires :

|  |  |
| --- | --- |
| 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 | */\*\**  *\* 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*](https://www.carnetdumaker.net/snippets/dossiers/2/) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

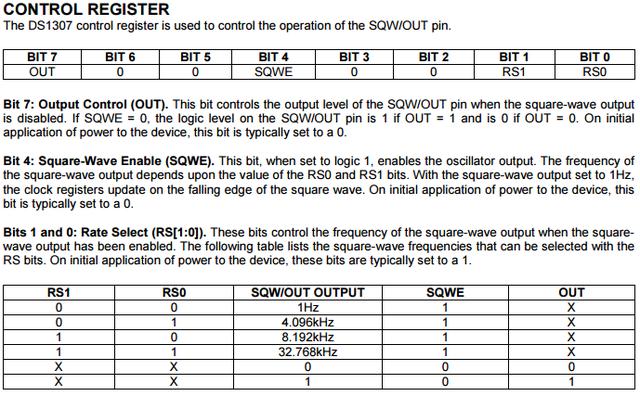
### Bonus : Utiliser la sortie programmable du module

[](https://www.carnetdumaker.net/images/description-de-la-broche-sqw-du-module-ds1307/)

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.

[](https://www.carnetdumaker.net/images/description-du-registre-de-controle-de-la-broche-sqw-du-module-ds1307/)

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) :

|  |  |
| --- | --- |
| 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 | */\*\**  *\* 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*](https://www.carnetdumaker.net/snippets/dossiers/3/) (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.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | */\*\* 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*    */\* Lit un octet depuis la mémoire du module RTC \*/*  Wire.requestFrom(DS1307\_ADDRESS, (byte) 1);  **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  2  3  4  5  6  7  8  9  10  11  12  13 | */\*\* Fonction d'écriture de la mémoire non volatile du module RTC (56 octets maximum) \*/*  void write\_nvram\_memory(byte address, byte data) {    */\* N'écrit pas en dehors de la NVRAM \*/*  **if** (address > DS1307\_NVRAM\_SIZE)  **return**;    */\* Début de la transaction I2C \*/*  Wire.beginTransmission(DS1307\_ADDRESS);  Wire\_write(DS1307\_NVRAM\_BASE + address); *// Ecriture mémoire NVRAM*  Wire\_write(data);  Wire.endTransmission(); *// Fin de transaction I2C*  } |

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

|  |  |
| --- | --- |
| 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 | */\*\* Fonction setup() \*/*  void setup() {    */\* Initialise le port I2C \*/*  Serial.begin(115200);    */\* Initialise le port I2C \*/*  Wire.begin();    */\* Lit la valeur actuelle \*/*  byte value = read\_nvram\_memory(0);  Serial.print(F("Valeur actuelle : "));  Serial.println(value);    */\* Met à jour la valeur \*/*  Serial.print(F("Nouvelle valeur : "));  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 :

|  |  |
| --- | --- |
| 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 | */\*\**  *\* 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*    */\* Lit un octet depuis la mémoire du module RTC \*/*  Wire.requestFrom(DS1307\_ADDRESS, (byte) 1);  **return** Wire\_read();  }  */\*\* Fonction d'écriture de la mémoire non volatile du module RTC (56 octets maximum) \*/*  void write\_nvram\_memory(byte address, byte data) {    */\* N'écrit pas en dehors de la NVRAM \*/*  **if** (address > DS1307\_NVRAM\_SIZE)  **return**;    */\* Début de la transaction I2C \*/*  Wire.beginTransmission(DS1307\_ADDRESS);  Wire\_write(DS1307\_NVRAM\_BASE + address); *// Ecriture mémoire NVRAM*  Wire\_write(data);  Wire.endTransmission(); *// Fin de transaction I2C*  }  */\*\* Fonction setup() \*/*  void setup() {    */\* Initialise le port I2C \*/*  Serial.begin(115200);    */\* Initialise le port I2C \*/*  Wire.begin();    */\* Lit la valeur actuelle \*/*  byte value = read\_nvram\_memory(0);  Serial.print(F("Valeur actuelle : "));  Serial.println(value);    */\* Met à jour la valeur \*/*  Serial.print(F("Nouvelle valeur : "));  Serial.println(value + 1);  write\_nvram\_memory(0, value + 1);  }  */\*\* Fonction loop() \*/*  void loop() {  *// Rien à faire*  } |

L'extrait de code ci-dessus est disponible en téléchargement sur [*cette page*](https://www.carnetdumaker.net/snippets/dossiers/4/) (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](https://www.carnetdumaker.net/articles/fabriquer-un-systeme-dacquisition-de-mesures-datalogger-avec-une-carte-arduino-genuino/)