

Utiliser la communication série I2C (librairie Wire) avec Arduino.



Ateliers Arduino

par X. HINAULT

www.mon-club-elec.fr



Tous droits réservés – 2012.

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

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

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

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

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

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

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

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

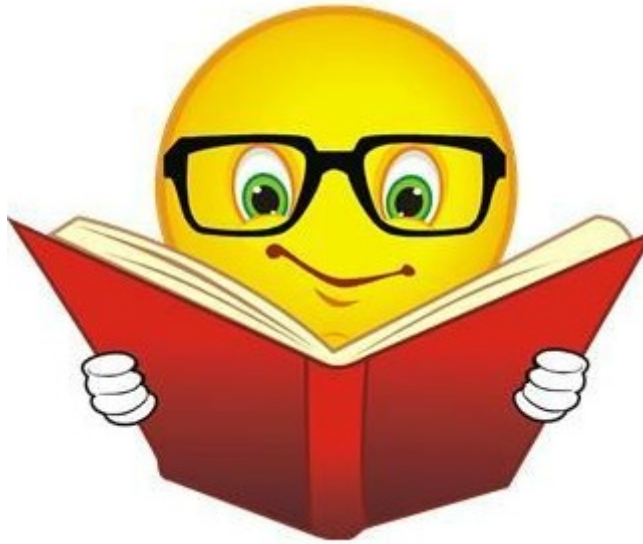
En mode fenêtre, activer le panneau latéral vous facilitera la navigation dans le document. Bonne lecture !

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

1. Intro

L'objectif ici est :

- de découvrir et d'apprendre à utiliser la communication I2C disponible avec Arduino,
 - de découvrir la librairie Wire du langage Arduino
 - à titre d'exemple, nous présenterons quelques composants I2C et le principe de leur utilisation,
 - vous découvrirez à titre indicatif quelques shields utilisant la communication I2C
- ... afin d'être en mesure d'utiliser le protocole de communication I2C avec Arduino.



Prêt ? C'est parti !

Pratique :

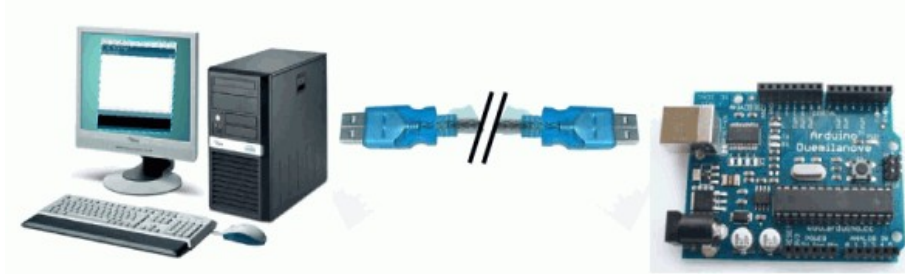
Les codes de cet atelier sont disponibles ici :

<https://github.com/sensor56/d8d16fa275fb7eb3c53073765b75e83b>

2. Matériel nécessaire pour les ateliers Arduino

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

De l'espace de développement Arduino

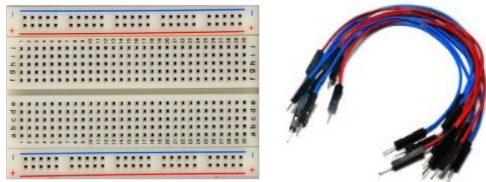


L'espace de développement Arduino associe :

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

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

Du nécessaire pour réaliser des montages sans soudure

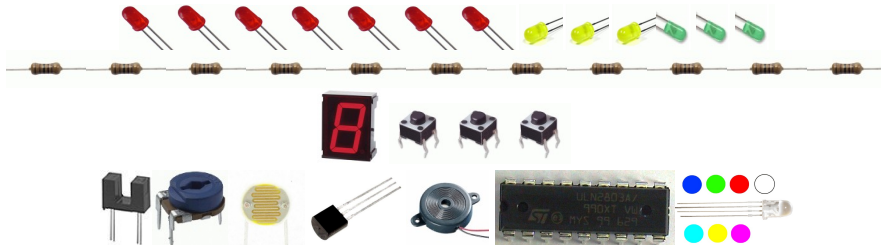


Pour réaliser des montages sans soudure, vous aurez besoin :

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

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

De quelques composants de base



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

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

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

GO TRONIC
ROBOTIQUE ET COMPOSANTS ÉLECTRONIQUES

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

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

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

3. Matériel spécifique nécessaire pour cet atelier

Pour cet atelier vous aurez besoin également d'un ou plusieurs dispositif I2C, par exemple :

L'étage horloge temps réel (« RTC ») I2C type DS1307 de la carte d'extension (shield) mémoire + temps réel (Snootlab)



La carte d'extension (ou shield) mémoire SD + « temps réel » est une carte électronique enfichable broche à broche sur la carte Arduino et qui dispose :

- d'un étage « carte mémoire SD » d'utiliser une carte mémoire micro SD, SD et SDHC. Cet étage utilise la **communication SPI** (broches 13,12,11, et 10) pour communiquer avec Arduino.
- d'un étage « temps-réel » basé sur un DS1307. Cet étage utilise la **communication I2C** (broches A4 et A5) pour communiquer avec Arduino.

disponible chez <http://snootlab.com/> Prix constaté : 18€ environ

Un shield LCD à communication I2C



DEULIGNE 1.1

Le [shield Deuligne de Snootlab](#) par exemple, a carte d'extension (ou shield) Arduino LCD à communication I2C :

- qui intègre notamment un mini-joystick,
- qui intègre un écran LCD alpha-numérique 2 lignes x 16 caractères : il est fourni avec une librairie dédiée qui supporte la plupart des instructions de la librairie Arduino **LiquidCrystal** native.

... et d'une manière générale, tout shield ou module I2C :



ROTOSHIELD

Shield de contrôle de 4 moteurs CC ou 2 moteurs pas à pas par I2C

4. Technique : Communication I2C : principe général

Intro

- L'interface I2C est un **protocole de communication série synchrone half-duplex** (bi-directionnel non simultané) utilisé par des dispositifs ou des microcontrôleurs (tels que la carte Arduino) **pour communiquer avec un ou plusieurs composants** périphériques rapidement sur de courtes distances.

Communication série synchrone halfduplex ???

En bon français, **série** ça veut dire que les données seront transmises à la « queue leuë » (les bits, çàd les 0 et les 1, les uns après les autres) sur un fil. **Synchrone**, ça veut dire que l'émetteur et le récepteur fonctionneront ensemble. **Half-duplex**, ça veut dire que les données transiteront dans les 2 sens sur un même fil, mais successivement (dans un sens ou l'autre à la fois).

- Ce protocole de communication est particulièrement pratique et facile à mettre en œuvre car il peut être utilisé avec plusieurs shields/capteurs différents au besoin.

Petit historique

La norme I2C (Inter Integrated Circuit) est une norme développée par Philips en 1982. Actuellement le protocole est à sa version 4.0 (2012).

Rapide comparatif SPI et I2C

- Le point commun entre les 2 protocoles de communication : c'est une communication série.
- Le protocole I2C permet de communiquer avec de nombreux dispositifs sur seulement 2 fils contre 3 + n avec la communication SPI (où n est le nombre de dispositifs).

Les éléments clés de la communication I2C

- Comme pour SPI, la première chose c'est que nous allons avoir 2 éléments qui vont « parler » entre eux :
 - celui qui aura l'initiative sera appelé le « maître »
 - celui qui fera ce qu'on lui dit sera appelé « l'esclave »
- Ensuite, les données vont transiter sur 1 fil comme on l'a dit pour les 2 sens de communication : le même fil servira pour communiquer du maître vers l'esclave ou de l'esclave vers le maître (1 sens à la fois)
- Enfin, les 2 éléments devront être synchronisés entre eux, ce qui se fera à l'aide d'un 3ème fil chargé de donner la cadence, appelé « horloge »

Retenez l'essentiel :

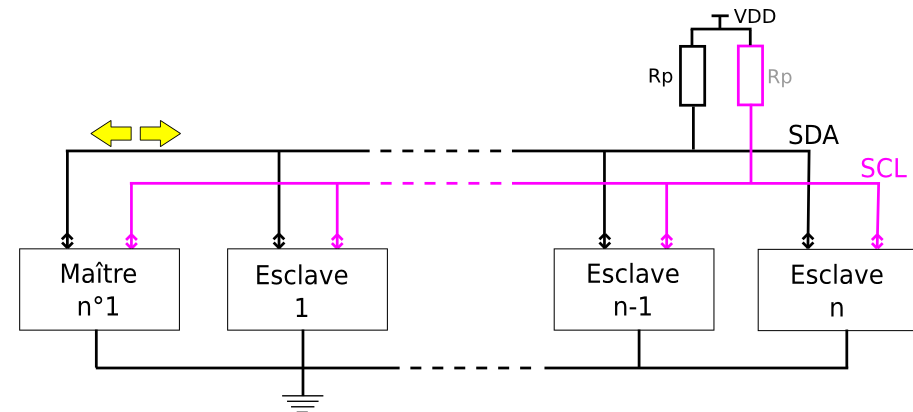
2 éléments qui discutent entre eux : l'esclave et le maître
1 fil de communication bidirectionnel
1 fil de synchronisation : le fil d' horloge

La même chose... en plus technique !

- Dans ce qui suit, on va redire exactement la même chose, mais en utilisant les mots techniques d'usage. Ne vous laissez pas impressionné, c'est simple en fait !
- Avec une connexion I2C, il y a toujours un composant maître (habituellement un microcontrôleur, Arduino pour nous) lequel commande-le ou les composants périphériques.

Selon les cas, Arduino sera utilisé en maître ou en esclave, ça dépendra.

- La connexion entre les éléments se fait avec :
 - une ligne de communication bidirectionnelle (appelée **SDA**)
 - une ligne d'horloge (appelée **SCL**)

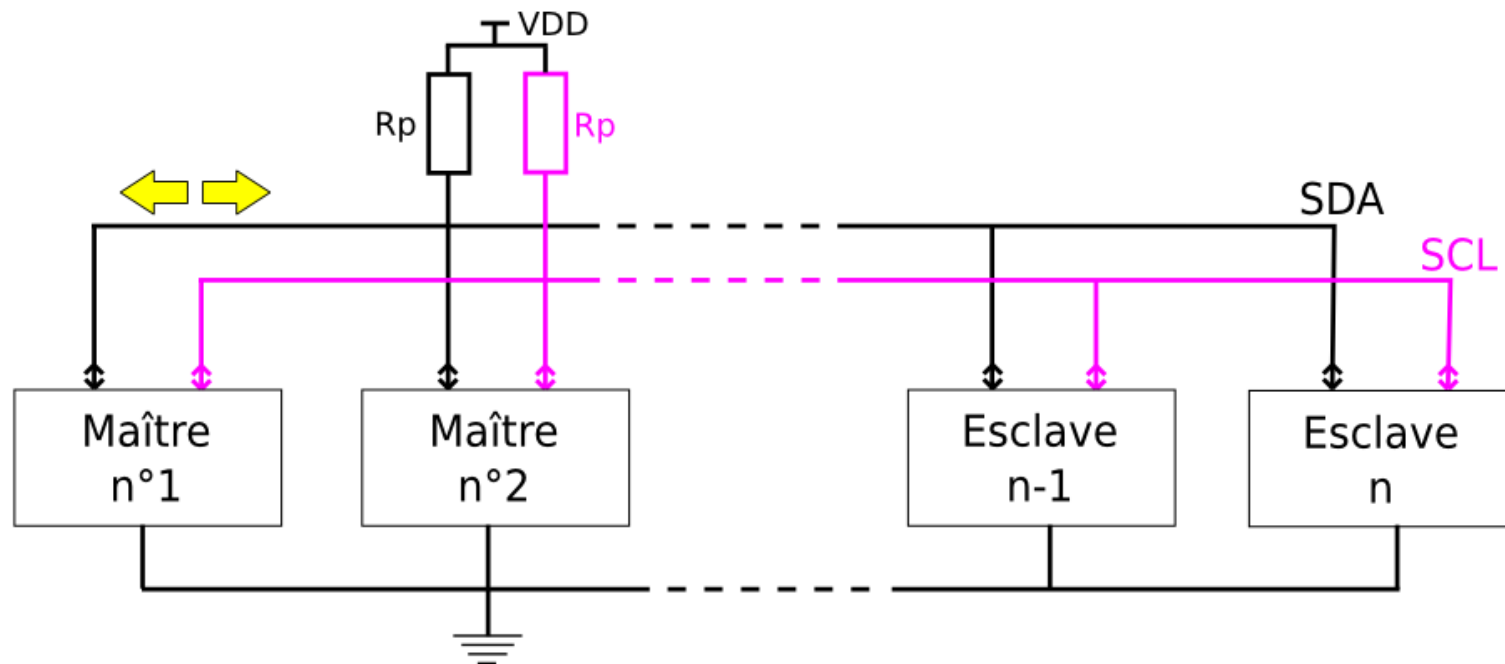


Technique : remarquer que les 2 lignes sont « rappelées au plus ».

- La question qui se pose : comment le maître identifie les différents « esclave » ?** En fait, le protocole I2C est assez élaboré et permet d'attribuer à chaque élément une **adresse** (un numéro identifiant) qui permettra à l'esclave concerné de savoir que le maître lui parle et aux autres esclaves, d'ignorer les données.

5. Pour info : topologie I2C multi-maîtres / multi-esclaves

- Il est même possible d'avoir un réseau I2C multi-maîtres, multi-esclaves (ceci est rendu possible en jouant sur les niveaux HAUT/BAS des maîtres : seul le maître qui parle sera au niveau bas)



Je vous donne l'info pour que vous le sachiez, même si en pratique, ce n'est pas ce que nous ferons ici.



6. Pour info : le protocole I2C

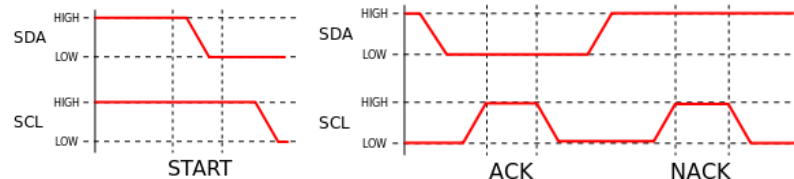
La communication I2C, comment ça marche ?

- Une communication I2C va typiquement se passer de la façon suivante :
 - le maître va envoyer l'**adresse** de l'esclave à qui il veut parler
 - suivi des **données** qui seront :
 - le code d'une instruction que l'esclave est capable de « comprendre »
 - l'adresse interne des « registres » (= octet mémoire interne) dans lesquels écrire/lire
 - les données à écrire
 - puis l'esclave concerné enverra sa réponse au maître...

Tous les détails de fonctionnement interne d'un dispositif I2C sont expliqués dans sa notice technique (ou datasheet)

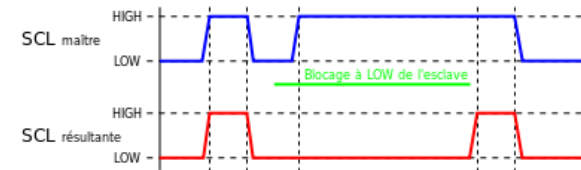
Les rouages du protocole I2C : 1. du maître vers l'esclave

- Le protocole I2C arrive à faire tout ça avec seulement 2 fils : c'est donc particulièrement astucieux... mais un poil complexe ! Du moins pour le détail, car en pratique, nous utiliserons une librairie du langage Arduino qui une fois de plus nous simplifiera la vie !
- La première étape : le maître annonce qu'il va parler en envoyant un signal de « Start »,
- suivi de l'émission du ou des octets d'adresse de l'esclave contacté : l'octet d'adresse contient 7 bits d'adresse + 1 bit R/W qui indique le sens de la communication. Ici, le bit R/W est à 0 : le maître « écrit ».

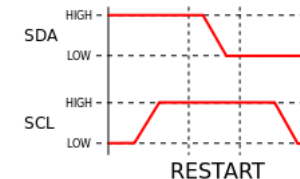


- A ce moment là, l'esclave doit renvoyer un signal « ACK » (si disponible) ou « NACK » (si indisponible) :
- Si le maître a bien reçu le signal ACK, il « envoie la sauce »... et envoie les octets de données voulus. L'esclave envoie un ACK entre chaque octet bien reçu.

- L'esclave peut demander une PAUSE si il le souhaite.



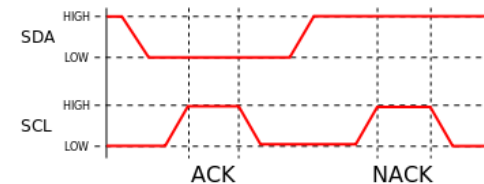
- Puis, le maître envoie un signal RESTART



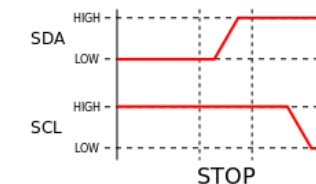
- Ensuite, le maître envoie un nouvel octet d'adresse avec cette fois le bit R/W est à 1 : le maître « écoute ».

Les rouages du protocole I2C : 2. de l'esclave vers le maître

- A ce moment là, l'esclave envoie les octets de réponse, et le maître répond à chaque fois par un ACK ou NACK



- Une fois l'envoi terminé, le maître envoie un signal de STOP.

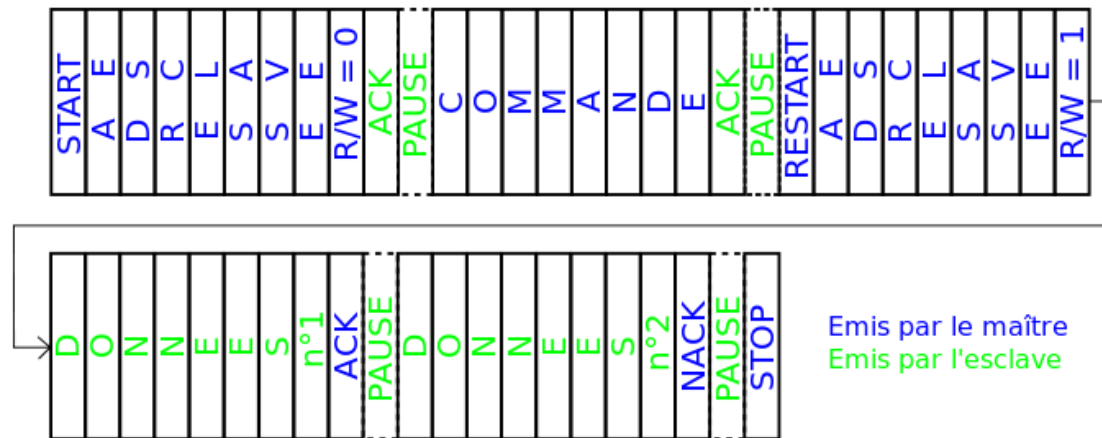


« Où là là... J'ai rien compris !!! C'est quoi ce délire ?? »

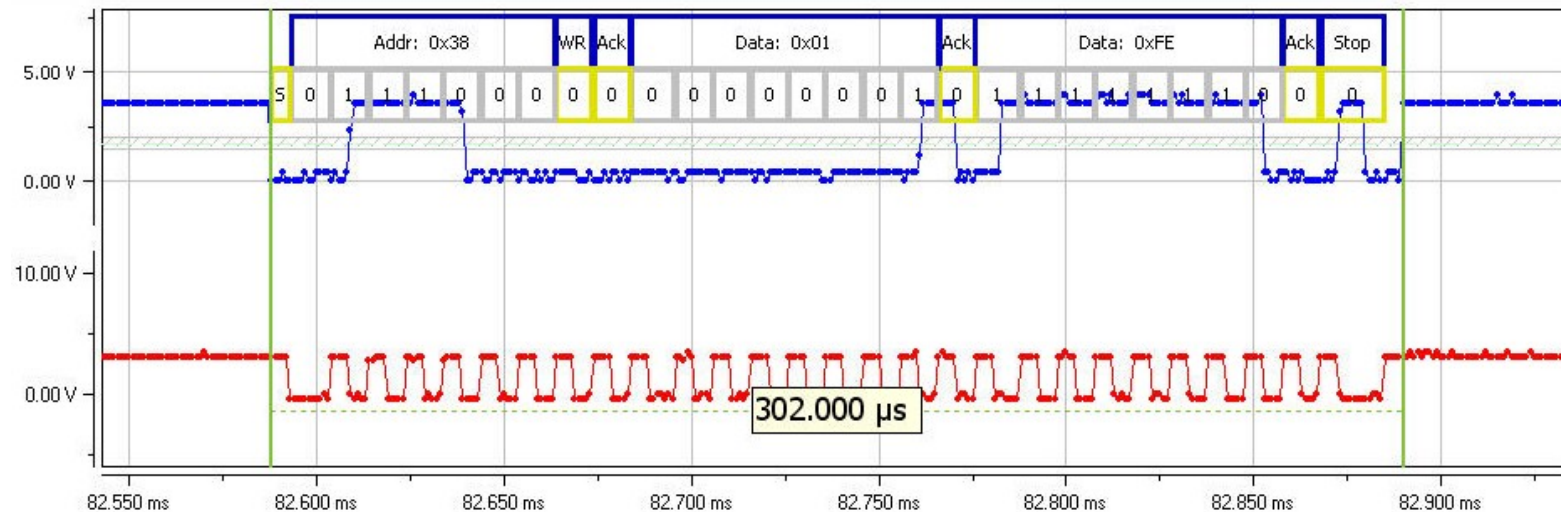
Pas de panique, c'est juste pour info, histoire que vous sachiez ce qui se passe quand vous allez utiliser I2C... mais en pratique, tout ceci va se faire automatiquement, sans vous ayez à vous en occuper !

7. Exemple d'enregistrement d'une trame de communication I2C

- En résumant ça dans un schéma, ça nous donne :



- Pour conclure la partie « technique » concernant l'I2C, voici un enregistrement d'une communication I2C : tout ce qui se passe sur les lignes **SDA** et **SCL**.



Source utile : <http://fr.wikipedia.org/wiki/I2C>

8. Exemple de dispositifs fonctionnant en I2C

- Les dispositifs utilisant I2C sont très nombreux et les shields Arduino utilisant I2C également, notamment :

Des dispositifs utilisant I2C

- Les capteurs utilisant I2C sont nombreux :
 - thermomètre, baromètre (BMP085), accéléromètre, etc...
- D'autres composants très pratiques utilisant I2C sont également nombreux, notamment : DS1307, circuit intégré implémentant l'heure « temps-réel »

Des shields Arduino utilisant I2C :

- Les shields Arduino utilisant I2C sont légion... ! De très nombreux fabricants en proposent : Adafruit, DFRobot, etc..
- A titre d'exemple, de nombreux shields de la gamme du fabricant français <http://snootlab.com/> notamment utilisent la communication I2C : contrôle PWM, contrôle de rotation de 2 moteurs pas à pas, étage « temps-réel » basé sur un DS1307, contrôle d'un afficheur LCD



I2C PWM DRIVER



I2C POWER PROTOSHIELD V2



ROTOSHIELD



MÉMOIRE



DEULIGNE 1.1

Des matériels existants utilisant la communication I2C

- On pourra citer notamment la manette wiiChuck, facile à utiliser à l'aide d'un simple adaptateur.





Remarque générale

Je constate que de nombreux utilisateurs Arduino sont « emballés » par les capteurs I2C et les utilisent en priorité, alors qu'ils débutent ou n'ont pas les bases.

Je tiens à redire ici qu'à mon sens, ce type de capteurs et dispositifs I2C ne sont pas adaptés pour l'initiation, et ce pour plusieurs raisons :

ils impliquent une complexité de fonctionnement sous-jacente et l'utilisation de bibliothèques spécifiques qui rendront la résolution de bugs assez vite complexe, ils laissent croire que l'on « sait faire », alors qu'au moindre souci, on ne saura pas trouver ce qui ne va pas, à moins d'utiliser un oscilloscope numérique... ils sont également souvent spécifiques d'un fabricant ou nécessitent une bibliothèque dédiée, ce qui n'est pas une bonne approche en phase d'apprentissage...

Je conseille fortement, dans une phase d'initiation et d'approche, d'utiliser en priorité de simples capteurs analogiques par exemple qui seront tout aussi efficaces, mais surtout, qui permettront de comprendre ce que l'on fait, de résoudre facilement les problèmes éventuels, d'adapter le code...

Utiliser I2C ?

Oui, mais seulement une fois que l'on est à l'aise et habitué avec Arduino et ses concepts de base...

... au risque d'une « illusion de savoir faire » et d'une incapacité à résoudre soi-même les difficultés rencontrées.

Ceci étant, l'utilisation des dispositifs I2C est particulièrement intéressante, et même indispensable, dans de nombreuses situations,

à réserver cependant aux utilisateurs ayant déjà une bonne expérience avec Arduino, ce qui sera votre cas très prochainement en utilisant les nombreux tutoriels que je vous propose !

9. Rappel : Langage Arduino : Introduction aux bibliothèques

C'est quoi une bibliothèque Arduino ?

Le langage Arduino comporte de nombreuses instructions comme vous avez pu le constater, une quarantaine en tout. Ces instructions sont intégrées dans ce que l'on appelle le « noyau » ou « cœur » (core en Anglais) du langage Arduino. Ces instructions sont « générales » et servent souvent.

Le langage Arduino peut cependant être étendu à la demande avec des instructions dédiées à certaines applications particulières : afin de ne pas surcharger inutilement le « cœur », ces instructions spécifiques ont été intégrées dans des « paquets d'instructions » appelés bibliothèques.

Comment ça marche ?

Par exemple, si on utilise un afficheur LCD, un servomoteur ou encore si l'on utilise un shield ethernet (réseau), on va intégrer dans notre programme la bibliothèque dédiée correspondante.

Principe général d'utilisation

Pour intégrer une bibliothèque dans un programme Arduino, c'est très simple : il suffit d'ajouter en début de programme une ligne de la forme :

```
#include <nombibliotheque.h> // bibliotheque pour servomoteur
```

ATTENTION : l'instruction include est un peu particulière : la ligne commence par un # et il n'y a pas de point virgule de fin de ligne !

Ensuite, dans le code, au niveau de l'entête déclarative, là où vous déclarez vos variables, il va falloir déclarer un objet (une sorte de super variable) représentant la bibliothèque. Cet objet est en fait une instance (= un exemplaire) d'une Classe (=le moule) qui regroupe les fonctions de la bibliothèque. On a :

```
ClasseObjet monObjet; // declare un objet
```

Généralement ensuite :

- au niveau de la fonction `setup()`, on initialise l'objet avec les paramètres voulus
- au niveau de la fonction `draw()`, on appelle les fonctions de la bibliothèque sous la forme que vous connaissez déjà :

```
monObjet.fonction( param, param, ..);
```

Rappel : pour utiliser une fonction d'une classe du langage Arduino, on utilise le nom de la classe + un point + le nom de la fonction.

Il peut exister des variantes selon les bibliothèques, mais grosso-modo, ça fonctionne de cette façon pour la plupart des bibliothèques Arduino.

Vous avez déjà utilisé une bibliothèque !

Si vous êtes attentifs à tout ce qu'on a déjà vu, vous me direz que ça ressemble étrangement à l'utilisation de la classe **Serial...** et vous aurez raison ! En fait, la classe Serial est une bibliothèque qui est intégrée implicitement lorsque vous lancez Arduino : c'est pour ça que vous n'avez pas besoin d'utiliser **#include** pour l'utiliser.

Les bibliothèques standards Arduino

Les bibliothèques Arduino disponibles sont nombreuses, et disposent chacune de quelques fonctions à plusieurs dizaines... ce qui étend considérablement la puissance du langage Arduino et qui en fait aussi tout son intérêt. Voici la liste des bibliothèques standards du langage Arduino (**le logiciel donne la liste...**) :

- [La bibliothèque Serial](#) - pour les communications séries entre la carte Arduino et l'ordinateur ou d'autres composants
- [La bibliothèque LCD](#) - pour l'utilisation et le contrôle d'un afficheur LCD alphanumérique standard.
- [La bibliothèque Servo](#) - pour contrôler les servomoteurs.
- [La bibliothèque Stepper](#) - pour contrôler les moteurs pas à pas (nécessite une interface de commande)
- [La bibliothèque Ethernet](#) - pour se connecter à Internet en utilisant le module Arduino Ethernet
- [La bibliothèque EEPROM](#) - référence - pour lire et écrire dans la mémoire EEPROM non volatile.
- [La bibliothèque SD](#) - référence - pour utiliser une carte mémoire SD (utiliser des fichiers, stocker des données, ...)
- [La bibliothèque SoftwareSerial \(Série Logicielle\)](#) - référence - pour communication série logicielle sur n'importe quelles broches de la carte Arduino
- [La bibliothèque Wire / I2C](#) - référence - Interface "deux fils" (TWI/I2C) pour envoyer et recevoir des données sur un réseau de modules ou capteurs.
- [La bibliothèque SPI \(Serial Peripheral Interface\)](#) - pour communication série avec des modules externes supportant le protocole SPI
- [Firmata](#) - pour communiquer avec des applications sur l'ordinateur utilisant un protocole série standard.

En jaune les plus utiles. Impressionnant non ? On les étudiera pas à pas...

Les bibliothèques de la communauté

A côté de ces bibliothèques standards, il existe toute une série de bibliothèques proposées par les uns et les autres et qui concernent des matériels spécifiques, ou autre. Par exemple :

- [La bibliothèque Keypad](#) - pour l'utilisation des claviers matriciels. (**hors référence**)

Faites un tour ici pour voir ce qui existe : <http://arduino.cc/playground/Main/LibraryList>

10. Langage Arduino : la librairie **Wire** pour l'utilisation de la communication série I2C

Présentation

- La librairie **Wire** met à disposition toutes les fonctions utilisées pour réaliser une communication série half-duplex I2C avec Arduino configuré aussi bien en esclave qu'en maître.

Inclusion

La librairie s'intègre dans un programme avec la ligne (pas de ; !!) :

```
#include <Wire.h> // inclut la librairie Wire pour communication I2C
```

Le constructeur de la classe

Le constructeur de la classe n'a pas besoin d'être déclaré et est **implicite** lors de l'inclusion de la librairie. Comme pour Serial, les fonctions seront directement accessibles sous la forme :

```
Wire.fonction() ; // appel type d'une fonction SPI
```

Les fonctions de la librairie

La librairie dispose de nombreuses fonctions, à savoir :

Fonctions d'initialisation

- begin()** : initialise communication avec Arduino "maître"
- begin(adresse)** : initialise communication avec Arduino "esclave"

Fonctions "mode maître"

- requestFrom(adresse, quantite)** : demande de données à un esclave
- beginTransaction(adresse)** : débute communication avec un esclave (ouvre stockage données à envoyer avec **write()**)
- endTransmission()** : envoi des données vers esclave
- write()** : écrit les données à envoyer vers esclave
- available()** : test si données disponibles en provenance esclave (cf **requestFrom()**)
- read()** : lit les données en provenance de l'esclave

Fonctions "mode esclave"

- write()** : envoie les données vers le maître après requête
- available()** : test si données disponibles en provenance du maître (cf **onReceive()**)
- read()** : lit données en provenance maître
- onReceive(fonction)** : définit la fonction à appeler sur réception de données en provenance du maître
- onRequest(fonction)** : définit la fonction à appeler sur requête du maître

Pour le détail complet des fonctions de la librairie, voir :

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

Principe d'utilisation en mode « Arduino maître »

La structure type d'un programme utilisant un dispositif I2C va être la suivante :

Au niveau de l'entête déclarative

- Inclusion de la librairie **Wire**

Au niveau de la fonction **setup()**

- Initialisation de l'afficheur avec la fonction **Wire.begin()**

Au niveau de la fonction **loop()**

- On commencera une transmission avec la fonction **beginTransaction(adresse)**
 - puis ajout d'une valeur à envoyer avec **write()**
 - et enfin envoi des valeurs avec **endTransmission()**
- On commencera une requête avec la fonction **requestFrom(adresse)**
 - on testera la réponse avec **available()**
 - et on lira les données disponibles avec **read()**

Exemple

```
#include <Wire.h>

byte val = 0;

void setup()
{
    Wire.begin(); // initialise I2C – Arduino maître
}

void loop()
{
    Wire.beginTransaction(44); // débute transmission vers #44 (0x2c)
    // l'adresse est donnée dans la doc du dispositif I2C utilisé

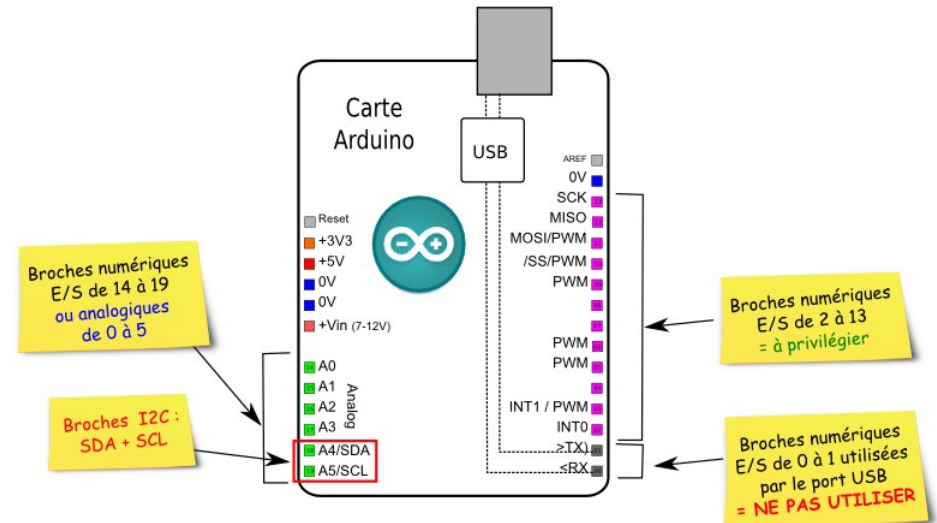
    Wire.write(val);           // envoie la valeur
    Wire.endTransmission();    // finalise la transmission

    val++;                     // incrémente valeur
    if(val == 64) // RAZ valeur de 64 à 0
    {
        val = 0;
    }
    delay(500); // pause entre 2 envois I2C
}
```

11. Connexions I2C de la carte Arduino

Les broches Arduino pour la communication I2C

- Très logiquement, Arduino va disposer de 2 broches pour réaliser une communication I2C :
 - une ligne de communication bidirectionnelle (appelée **SDA**)
 - une ligne d'horloge (appelée **SCL**)
- Sur la carte Arduino **UNO**, Duemilanove et autres cartes basées sur les ATmega 168 / 328, le bus I2C utilise les broches :
 - **A4 (SDA)**,
 - **A5 (SCL)**
- Sur la Leonardo (**Attention : pas pareil que la UNO !**) :
 - 2 (SDA),
 - 3 (SCL)
- Sur l'Arduino Mega, ce sont les broches :
 - 20 (SDA),
 - 21 (SCL)



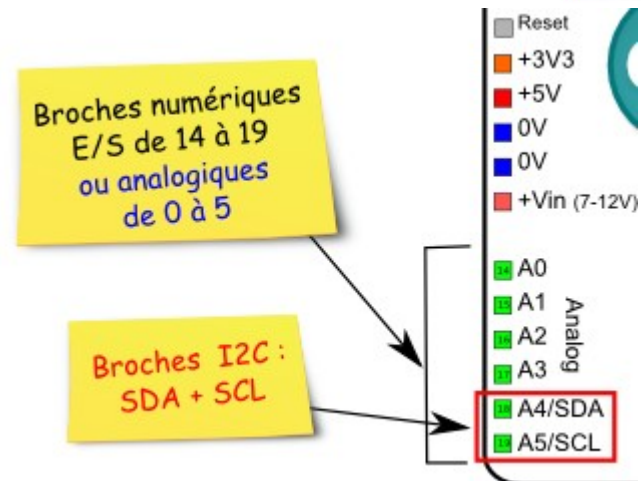
Remarque : SPI et I2C

Comparativement à une communication SPI, **la communication I2C n'utilise que 2 broches**, et ce, même si plusieurs dispositifs sont utilisés ! La communication SPI utilisera quant à elle 3 broches communes + 1 broche de sélection par dispositif.

Ainsi, pour 6 dispositifs :
avec I2C : utilisation de 2 broches seulement !
avec SPI : utilisation de 3 + 6 = 9 broches...

Donc, lorsque la vitesse n'est pas critique, **utiliser des dispositifs I2C est une excellente solution pour simplifier les projets utilisant de nombreux modules, capteurs, etc...**

Enfin, I2C et SPI peuvent être utilisés simultanément !

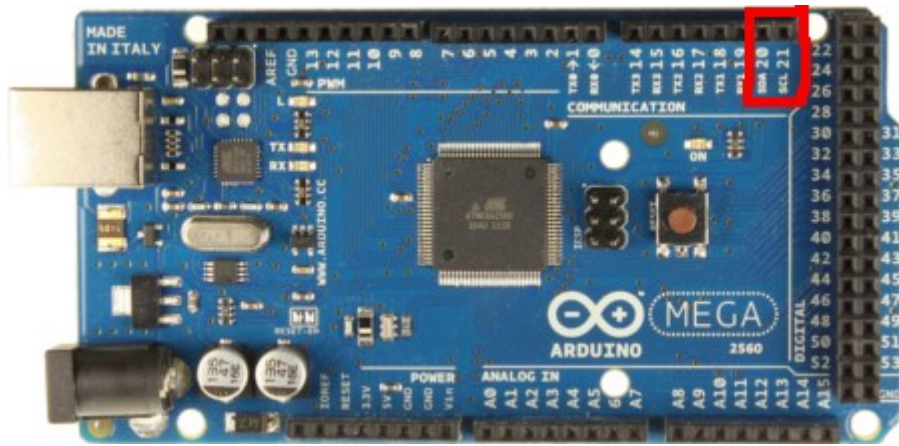


Remarquer que l'utilisation de la communication I2C fait « perdre » 2 broches analogiques sur la UNO/Duemilanove.

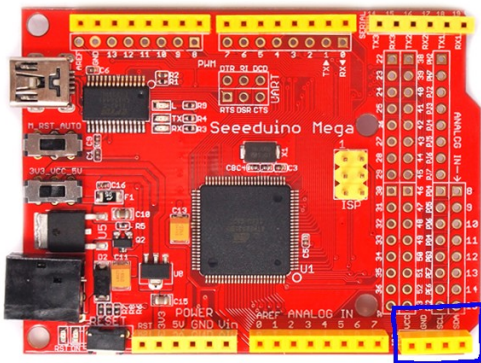
12. Truc technique : utiliser un dispositif ou un shield I2C avec une carte Arduino Mega

Les connexions I2C de la Arduino Mega

- Attention : le brochage de la connexion I2C n'est pas la même sur la carte Mega (ni sur la Leonardo d'ailleurs) que sur la carte UNO !
- Voici le brochage I2C des différentes cartes Arduino :
 - Uno, Ethernet : A4 (SDA), A5 (SCL)
 - Mega2560 : 20 (SDA), 21 (SCL)
 - Leonardo : 2 (SDA), 3 (SCL)
 - Due : 20 (SDA), 21 (SCL), SDA1, SCL1

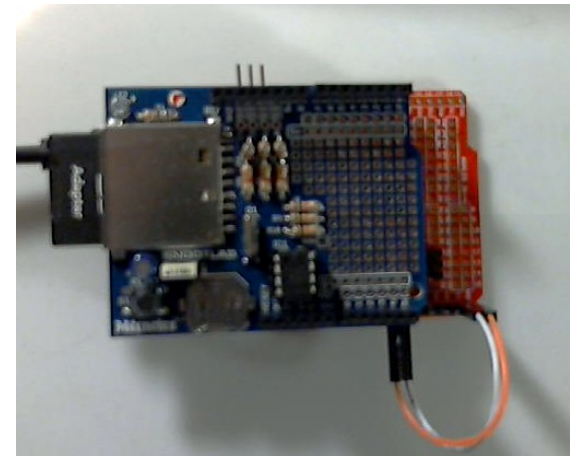


Noter que la localisation des broches 20/21 peut varier selon les modèles de carte Arduino Mega. Par exemple, la Mega Seedeino dispose d'un connecteur I2C séparé :



J'ai un shield I2C pour une Arduino UNO... comment je fais avec une Arduino Mega ??

- Dans ce cas, la « combine » consiste à « plier » latéralement les pattes A4 et A5 (SLA, SCL) du shield utilisé et à les connecter par des jumpers aux broches 20 et 21 (SLA et SCL) de la carte Arduino Mega.
- Et en faisant cela pour le premier shield enfiché sur la carte, on pourra empiler d'autres shields utilisant I2C par dessus !



Exemple de reprise des broches I2C 20 (SDA) et 21 (SCL) d'une Mega Seedeino sur les broches A4 (SDA) et A5(SCL) d'un shield Arduino.



13. Informations techniques importantes :

I2C et les interruptions

- La librairie **Wire.h** utilise une interruption interne (TWI), notamment la fonction `endTransmission()` pour détecter la fin de la transmission.
- **De plus cette interruption est bloquante** : ceci veut dire que tant qu'une communication I2C n'est pas correctement terminée, le programme est stoppé !

En pratique, à retenir :

Ne jamais désactiver les interruptions dans un programme utilisant la librairie Wire.h

Autrement dit, pas d'instructions `cli()` ou `noInterrupts()` dans un programme Arduino utilisant la librairie Wire.h

I2C et nombre de dispositifs

- Selon les dispositifs utilisés, il est possible d'utiliser jusqu'à 64 voire 128 dispositifs sur un même bus I2C, ce qui laisse de la marge ! Et le tout sur 2 lignes de communication seulement, auxquelles il faut ajouter la masse et le 5V.

Débit de communication

- Le débit de base est de l'ordre de 100 à 400 Kbits/secondes soit 20 KOctets/s et plus.

14. Info : Exemple de composant I2C : le Max7313, expandeur I2C PWM.

Intro

- Je vous propose, à titre d'illustration, et sans rentrer dans des détails trop techniques, de découvrir un composant I2C.
- J'ai choisi de vous présenter un circuit qui est utilisé dans certains shields Arduino I2C, le **Max7313**. Il s'agit d'un expandeur I2C qui permet d'ajouter 16 voies E/S PWM à l'Arduino à partir des 2 broches I2C uniquement (comme quoi l'Arduino Mega n'est pas la seule solution au manque de broches!).
- Ce type de circuit est pratique notamment pour contrôler de nombreuses LED RGB notamment. Chaque voie PWM dispose de 16 niveaux.
- Ce circuit se présente dans un boîtier DIL 18. Son brochage n'a pas grande importance ici. Je vais surtout m'attacher à vous expliquer sa logique interne de communication I2C.

Principe de fonctionnement interne

- Un composant I2C est un composant élaboré qui intègre toute une logique permettant de « comprendre » les instructions qu'il reçoit par le bus I2C et de stocker des paramétrages.
- Dans notre cas, le Max7313 va disposer d'octets de stockage (appelés « registres ») ayant des fonctions précises :
 - registres de **configuration**
 - registres d'**état** des broches E/S
 - registres de **paramétrage** des impulsions PWM
 - ...

Principe d'une écriture I2C vers le Max7313

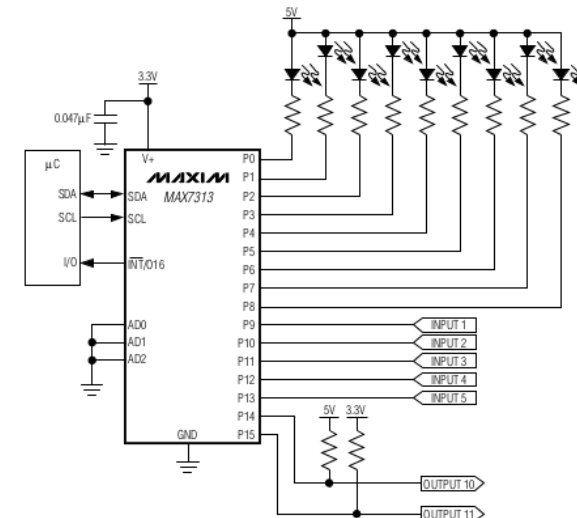
- Lorsque Arduino va communiquer en écriture avec le Max7313, il va successivement :
 - **envoyer l'adresse** attribuée au composant utilisé (fixée par la connexion de certaines broches du circuit),
 - envoyer ensuite **l'adresse du registre utilisé**
 - puis envoyer les **données à écrire** dans ce registre

Vous comprenez dès lors que la « cuisine » qui va être nécessaire pour prendre en main un composant donné ne va pas être simple : c'est pourquoi, la plupart du temps, une librairie va être fournie qui s'occupera de gérer la communication I2C comme nous le verrons par la suite...

Les « registres » du Max7313

REGISTER	ADDRESS CODE (HEX)
Read input ports P7–P0	0x00
Read input ports P15–P8	0x01
Blink phase 0 outputs P7–P0	0x02
Blink phase 0 outputs P15–P8	0x03
Ports configuration P7–P0	0x06
Ports configuration P15–P8	0x07
Blink phase 1 outputs P7–P0	0x0A
Blink phase 1 outputs P15–P8	0x0B
Master, O16 intensity	0x0E
Configuration	0x0F
Outputs intensity P1, P0	0x10
Outputs intensity P3, P2	0x11
Outputs intensity P5, P4	0x12
Outputs intensity P7, P6	0x13
Outputs intensity P9, P8	0x14
Outputs intensity P11, P10	0x15
Outputs intensity P13, P12	0x16
Outputs intensity P15, P14	0x17

Montage type



Remarque importante : distinguer protocole de communication et fonctionnement interne d'un dispositif.

Tout comme le protocole série SPI, le protocole de communication I2C est avant tout un protocole de communication.

Il faut bien comprendre que ce protocole de communication n'a rien à voir avec le fonctionnement interne d'un dispositif qui aura ses propres instructions internes, ses propres registres, etc... Il en découle que prendre en main « de zéro » un dispositif I2C suppose l'étude approfondie de la documentation du dispositif en question, notamment des instructions reconnues (appelées « opcode »), des adresses des registres internes, du format des données, etc...

Heureusement pour nous, encore une fois, les fabricants de shields utilisant I2C fournissent une librairie Arduino qui va donner accès aux fonctionnalités essentielles sans avoir à étudier tous les détails du fonctionnement interne d'un dispositif I2C.

Il en découle cependant plusieurs conséquences :

si vous n'étudiez pas le détail de la librairie, « vous perdrez la main » sur la bonne exécution d'un programme : ça passe ou ça ne passe pas... **Mais si ça ne marche pas, vous avez du boulot...** Donc choisir des dispositifs I2C pour lesquels existe une librairie Arduino, pour lesquels vous trouvez des exemples nombreux sur internet, pour lesquels le fabricant est réputé pour son support de qualité (Adafruit, Polulu, etc...) sinon éviter...

d'autre part, utiliser un dispositif I2C, c'est utiliser un matériel particulier spécifique à un constructeur de shields ou à un fabricant de circuits intégrés et par conséquent, il ne s'agit pas de connaissances polyvalentes et réutilisables : ce sont des connaissances spécifiques. Cela ne veut pas dire qu'il ne faut pas le faire... mais il faut en être conscient.

Exemple concret :

En utilisant un afficheur LCD standard alpha-numérique avec la librairie Arduino, vous serez en terrain « standard » et il sera facile de vous aider à dépanner... De plus, votre montage sera facilement reproductible et aura une durée de vie plus longue (un standard, généralement, ça dure longtemps...)

Par contre, si vous utilisez un afficheur LCD à communication I2C, vous devenez dépendant de la librairie spécifique du fabricant, du modèle utilisé... et vous ne trouverez de l'aide qu'auprès d'utilisateurs du même modèle ou du fabricant. Votre montage ne sera reproductible qu'avec ce modèle précis et aura une durée de vie plus courte (si le fabricant ne le fait plus, etc...) Encore une fois, ça ne veut pas dire qu'il ne faut pas le faire, mais simplement être conscient de la position dans laquelle on se place, bien peser les avantages potentiels... Il y a aussi des composants I2C qui sont devenus très répandus et donc polyvalents.

En résumé, privilégier les solutions standards en phase d'initiation, d'autant que c'est bien souvent nettement moins cher !

« Ok, bien compris... ! Mais bon, là c'est un tuto I2C non ? Alors, qu'est-ce qu'on attend pour passer à l'action ?? »

Justement, j'y arrive : je vous propose à présent quelques exemples d'utilisation d'un dispositif I2C avec la librairie fournie par le fabricant. Prêt ? C'est parti... !

15. Exemple I2C : le DS1307 : circuit intégré « Temps-réel » RTC (Real Time Clock) à communication I2C

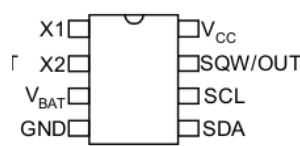
Présentation

- Le circuit DS1307 est un circuit intégré à **communication I2C**, qui assure le calcul automatique de la date, de l'heure, du jour et de l'année de façon fiable (utilise un quartz horloger).
- Typiquement, l'heure est fixée avec l'heure courante au moment de la programmation initiale.
- Ce circuit est typiquement alimenté par une pile bouton 3V qui assure son bon fonctionnement et la mémorisation de l'heure même lorsque le circuit est hors-tension.
- Un tel circuit « temps-réel » est très pratique pour réaliser notamment de l'enregistrement de données « horodatées ».

Le DS1703 utilisé sur de nombreux shields est facile à trouver et est très utilisé. Il dispose d'une librairie Arduino opérationnelle.

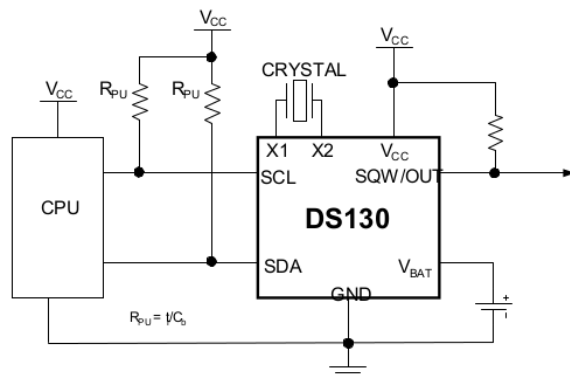
Le brochage

- Le DS1307 se présente sous la forme d'un boîtier DIL 8 broches :



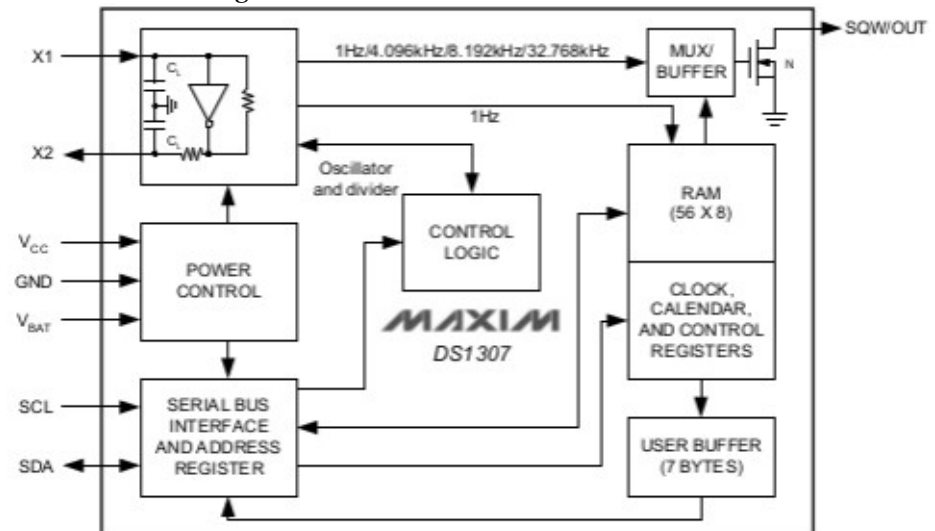
- Il dispose de 2 broches pour la communication I2C (SCL et SDA), de 2 broches pour le quartz (X1 et X2) ainsi qu'une pour la pile (VBat).

Le montage type



Détails internes du DS1307, circuit I2C

- Ce circuit assure le calcul automatique de l'heure, de la date, du jour, des secondes... Logiquement, en interne, il dispose :
 - d'un étage oscillateur,
 - de plusieurs registres qui contiennent les données d'heure et de date,
 - de l'étage de communication I2C...

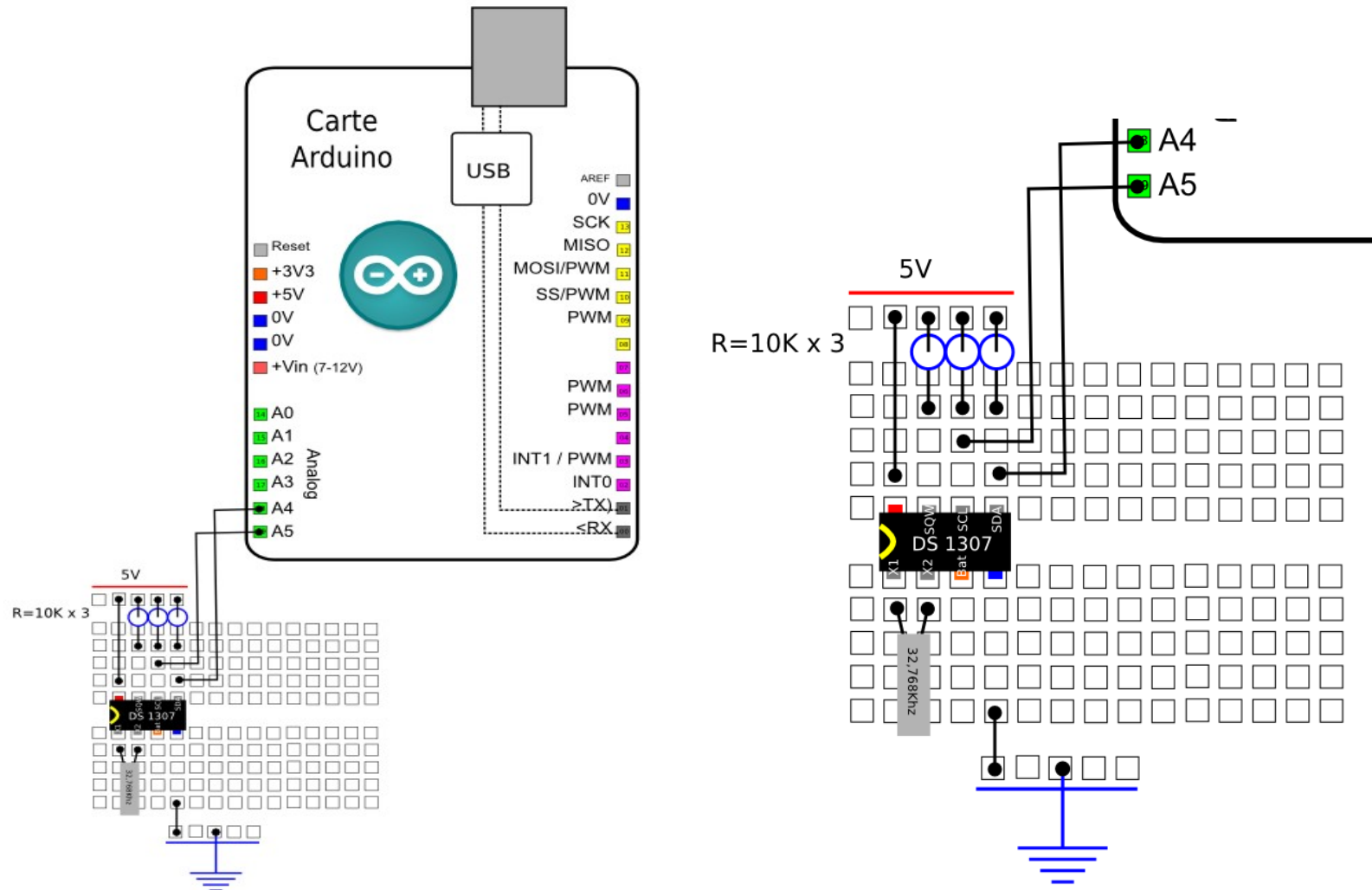


ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00-59
01h	0	10 Minutes			Minutes				Minutes	00-59
02h	0	12	10 Hour PM/AM	10 Hour	Hours				Hours	1-12 +AM/PM 00-23
		24								
03h	0	0	0	0	0	DAY			Day	01-07
04h	0	0	10 Date			Date			Date	01-31
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

- Je ne détaille pas ici le protocole de communication I2C qui sera géré par la librairie utilisée (voir ci-dessous).
- Pour plus de détails, se reporter au datasheet du DS1307.

16. Utiliser un DS1307 « brut » sur plaque d'essai

- Comme on vient de le voir, l'utilisation d'un DS1307 nécessite :
 - un quartz horloger à 32,768 KHz
 - de 3 résistances de rappel au plus, de 10KOhms typiquement
 - et... c'est tout.
- Le montage sur une plaque d'essai pour une utilisation avec Arduino sera le suivant :



17. Exemple de shield Arduino utilisant I2C : L'étage RTC du shield « Mémoire » de chez Snootlab

Présentation

La carte d'extension (ou shield) mémoire SD + « temps réel » est une carte électronique enfichable broche à broche sur la carte Arduino et qui dispose :

- d'un étage « carte mémoire SD » d'utiliser une carte mémoire micro SD, SD et SDHC. Cet étage utilise la **communication SPI** (broches 13,12,11, et 10) pour communiquer avec Arduino.
- d'un étage « temps-réel » basé sur un DS1307. Cet étage utilise la **communication I2C** (broches A4 et A5) pour communiquer avec Arduino.

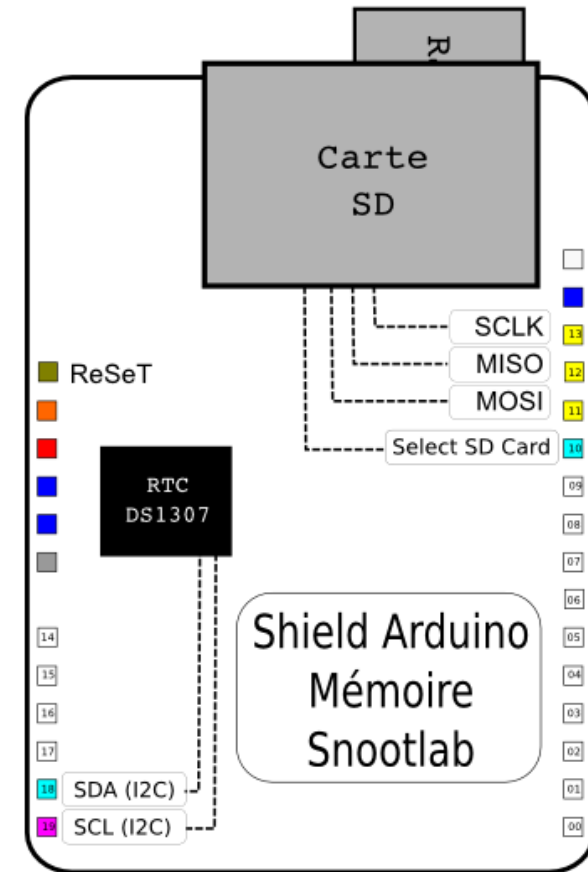
disponible chez <http://snootlab.com/> Prix constaté : 18€ environ



L'étage « temps-réel »

- L'étage de ce shield qui nous intéresse ici est l'étage « temps réel » à communication I2C.
- Cet étage « temps-réel » est basé sur un composant I2C, le DS1307, qui assure le calcul automatique de la date, de l'heure, du jour et de l'année de façon fiable (utilise un quartz horloger). Ce circuit est alimenté par une pile bouton qui assure son bon fonctionnement même lorsque le shield est hors-tension.
- Un tel étage « temps-réel » est très pratique pour réaliser notamment de l'enregistrement de données « horodatées », et il est donc logique de l'associer à un étage de stockage sur carte SD comme ici.

Le DS1703 utilisé sur ce shield est facile à trouver et est très utilisé, dispose d'une librairie Arduino opérationnelle. Plusieurs autres shields ou modules utilisent également ce circuit très polyvalent.



Quelque soit le shield ou module RTC que vous utiliserez, le principe sera le même : les broches SDA et SCL de votre shield / module seront connectées aux broches A4 (SDA) et A5 (SCL) de la carte Uno.

Sur la Mega et la Due, aux broches 21 (SCL) et 20 (SDA)
Sur la Leonardo, aux broches 2 (SDA) et 3 (SCL)

18. Exemple I2C : Temps-réel avec un DS1307 : la librairie RTCLib

La librairie Arduino RTCLib pour DS1307

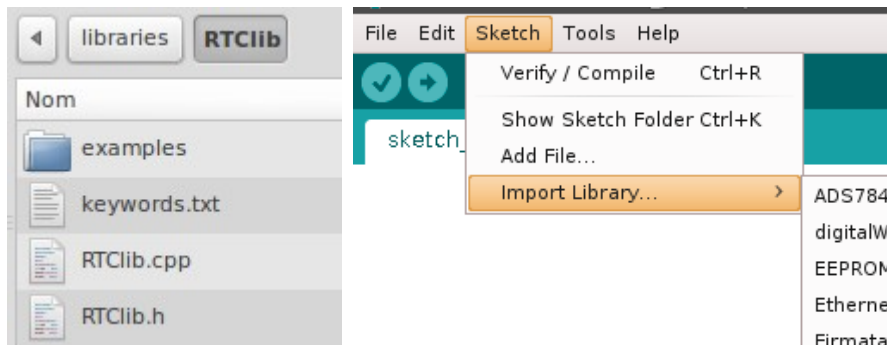
- La librairie RTCLib pour DS1307 implémente toutes les fonctions utiles pour l'utilisation du « temps-réel » avec ce circuit et également à partir de la base temps `millis()` de l'Arduino.

Télécharger la librairie

- La librairie RTCLib est disponible ici : <https://github.com/adafruit/RTCLib>

Installation

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



Le constructeur principal

- Trois constructeurs pour cette librairie :

```
DateTime now ; // représente une date/heure
RTC_DS1307 RTC ; // représente le DS1307
RTC_Millis RTC; // représente une base temps basée sur millis() !
```

Fonctions de la librairie

Des objets RTC

- `begin()` : initialise la base temps réel
- `adjust()` : règle la base temps réel
- `isrunning()` : test si la base temps réel est active
- `now()` : renvoie la date/heure courante

De l'objet DateTime

- `year()`, `month()`, `day()` : renvoie l'année, mois, jour
- `hour()`, `minute()`, `second()` : renvoie heure, minute, secondes
- `dayOfWeek()` : renvoie le jour de la semaine
- `secondstime()` : renvoie le temps absolu en secondes depuis le 1/1/2000
- `unixtime()` : renvoie le temps unix, le nombre de secondes depuis le 1/1/1970

Code d'exemple

```
#include <Wire.h>
#include "RTCLib.h"

RTC_DS1307 RTC; // déclare objet représentant le DS1307

void setup () {
    Serial.begin(115200); // initialise la communication série
    Wire.begin(); // initialise I2C
    RTC.begin(); // initialise le DS1307
}

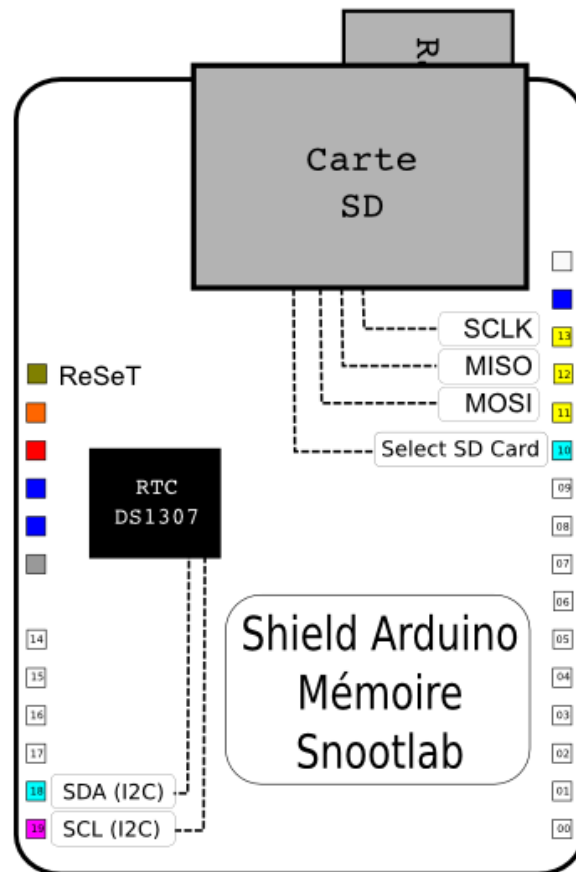
void loop () {
    DateTime now = RTC.now(); // récupère l'heure courante

    Serial.print(now.hour(), DEC); // affiche heure
    Serial.print(':');
    Serial.print(now.minute(), DEC); // affiche minutes
    Serial.print(':');
    Serial.print(now.second(), DEC); // affiche secondes
    Serial.println();
    delay(3000); // pause
}
```

19. Exemple I2C : Temps-réel avec un DS1307 : le montage

Dans ce premier exemple, nous allons voir comment une librairie dédiée permet de simplifier grandement l'utilisation d'un composant I2C : toute la communication I2C « directe » est gérée par la librairie et les fonctions de la librairie permettent d'accéder directement aux fonctionnalités utiles.

- Le montage se résume à sa plus simple expression : il suffit d'enficher le shield « Mémoire » sur la carte Arduino, broche à broche :



Ici, la carte mémoire SD n'est pas utilisée.

20. Exemple I2C : Temps-réel avec un DS1307 : le programme

Ce qu'on va faire ici...

- Ici, nous allons tout simplement tester l'affichage de l'heure et de la date avec le DS1307 en utilisant la communication I2C et la librairie RTCLib.

Entête déclarative

Inclusion des librairies utiles

- On commence par inclure les librairies
 - la librairie Wire.h pour la communication I2C
 - la librairie RTCLib.h pour la gestion simplifiée du temps réel avec le DS1307

Objet utiles

- On déclare un objet RTC_DS1307 représentant la base temps basée sur le DS1307 :

```
#include <Wire.h>
#include "RTCLib.h"

RTC_DS1307 RTC; // déclare objet représentant le DS1307
```

Fonction **setup()**

Initialisation Série

- On initialise la communication série à 115200 Bauds avec l'instruction **Serial.begin()**

Initialisation I2C

- On initialise la communication I2C en mode « Arduino maître » avec l'instruction **Wire.begin()**

Initialisation « temps-réel »

- On initialise le DS1307 à l'aide de la fonction **begin()** de l'objet RTC_DS1307 précédemment déclaré

```
void setup () {  
  Serial.begin(115200); // initialise la communicatin série  
  Wire.begin(); // initialise I2C  
  RTC.begin(); // initialise le DS1307  
}
```

Fonction **loop()**

- On commence par créer un objet DateTime représentant la date/heure actuelle
- Puis, par un jeu de Serial.print(), on affiche successivement l'année, le mois, le jour, l'heure, les minutes, les secondes...
- le code boucle après une pause d'une seconde

```
void loop () {  
    DateTime now = RTC.now(); // récupère l'heure courante  
  
    Serial.print(now.year(), DEC); // affiche année  
    Serial.print('/');  
    Serial.print(now.month(), DEC); // affiche mois  
    Serial.print('/');  
    Serial.print(now.day(), DEC); // affiche jour  
    Serial.print(' ');  
    Serial.print(now.hour(), DEC); // affiche heure  
    Serial.print(':');  
    Serial.print(now.minute(), DEC); // affiche minutes  
    Serial.print(':');  
    Serial.print(now.second(), DEC); // affiche secondes  
  
    Serial.println();  
    delay(1000); // pause  
}
```

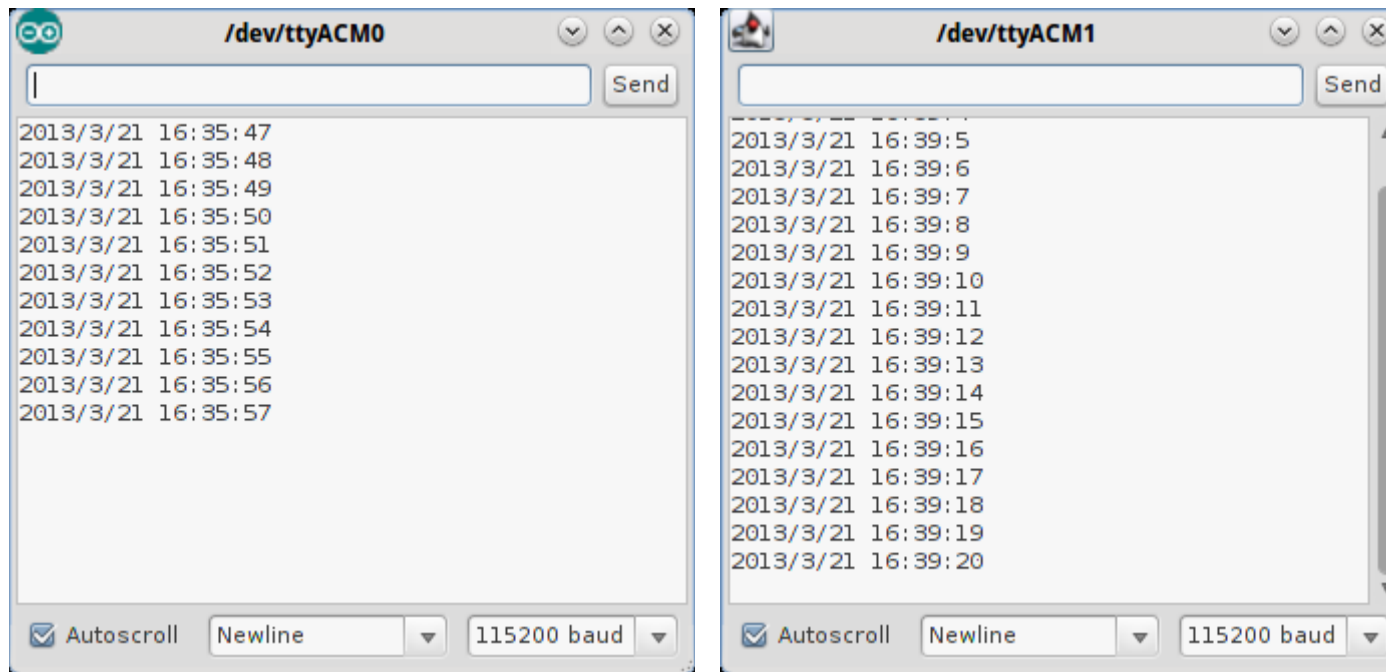
Vous allez me demander : « Mais comment le DS1307 se met-il à l'heure ? »

Et bien, très astucieusement, la librairie **RTCLib** va initialiser le **DS1307** avec l'heure du système de votre ordinateur au moment de la programmation.
Simple et efficace !

La fonction **adjust()** permet au besoin de régler l'heure en cours d'exécution.

Fonctionnement du programme

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



```
/dev/ttyACM0
2013/3/21 16:35:47
2013/3/21 16:35:48
2013/3/21 16:35:49
2013/3/21 16:35:50
2013/3/21 16:35:51
2013/3/21 16:35:52
2013/3/21 16:35:53
2013/3/21 16:35:54
2013/3/21 16:35:55
2013/3/21 16:35:56
2013/3/21 16:35:57

/dev/ttyACM1
2013/3/21 16:39:5
2013/3/21 16:39:6
2013/3/21 16:39:7
2013/3/21 16:39:8
2013/3/21 16:39:9
2013/3/21 16:39:10
2013/3/21 16:39:11
2013/3/21 16:39:12
2013/3/21 16:39:13
2013/3/21 16:39:14
2013/3/21 16:39:15
2013/3/21 16:39:16
2013/3/21 16:39:17
2013/3/21 16:39:18
2013/3/21 16:39:19
2013/3/21 16:39:20
```

Débranchez votre Arduino, patientez un moment puis rebranchez-là : Arduino est toujours l'heure !! Ceci grâce au DS1307 et à sa pile...

Simple et efficace non ?

Une solution polyvalente qui pourra être utilisée dans toutes les situations où il est nécessaire de gérer l'heure et la date réelle de façon précise et non volatile.



21. Exemple I2C : Utiliser un afficheur LCD à communication I2C : prérequis

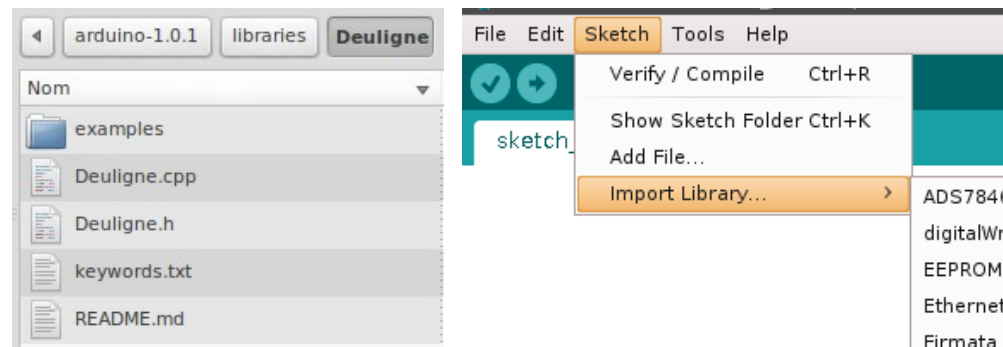
Un shield LCD à communication I2C

- Un shield LCD à communication I2C : le [shield Deuligne de Snootlab](#) par exemple, qui intègre notamment un mini-joystick.



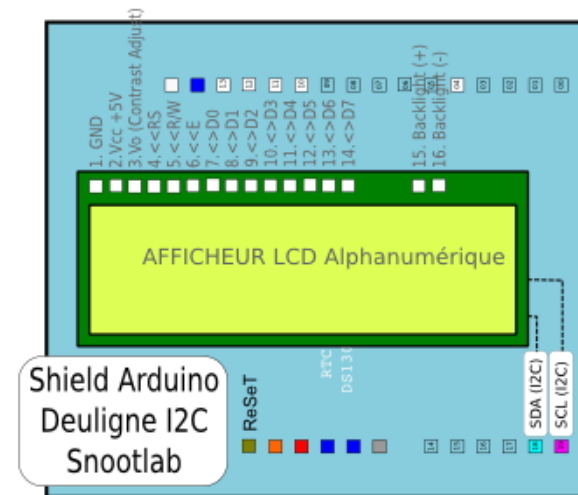
La librairie Arduino associée :

- La librairie du shield « Deuligne » est disponible ici : <https://github.com/Snootlab/Deuligne/tree/Arduino-1.0> (Veiller à choisir la branche 1.0)
- Télécharger l'archive, au format zip ou autre. L'extraire
- Vérifier que le nom du répertoire de la librairie est strictement le même que le nom du fichier *.h ou *.cpp principal. Corriger au besoin. Ici le nom est **Deuligne**
- Copier/coller le répertoire de la librairie dans le répertoire libraries de votre répertoire Arduino
- Relancer Arduino et vérifier que la librairie est présente dans le menu **Sketch > ImportLibrary**.



22. Exemple I2C : Utiliser un afficheur LCD à communication I2C : le montage

- Le montage est très simple et consiste à enficher broche à broche le shield « Deuligne » sur la carte Arduino : toutes les broches Arduino sont utilisables sauf la A4 (SDA) et la A5 (SCL).



23. Exemple I2C : Utiliser un afficheur LCD à communication I2C : le programme

Ce qu'on va faire ici...

- Ici, nous allons tout simplement afficher un message sur l'afficheur, un classique « Hello world ! »

Entête déclarative

Inclusion des librairies utiles

- On commence par inclure les librairies
 - la librairie Wire.h pour la communication I2C
 - la librairie Deuligne.h pour la gestion de l'afficheur LCD I2C

Objet utiles

- On déclare un objet Deuligne représentant l'afficheur :

```
//--- inclusion des librairies
#include "Wire.h" // librairie I2C
#include <Deuligne.h> // librairie afficheur I2C by Snootlab

Deuligne lcd; // déclare objet représentant l'afficheur LCD I2C
```

Fonction **setup()**

Initialisation I2C

- On initialise la communication I2C en mode « Arduino maître » avec l'instruction **Wire.begin()**
- On initialise l'afficheur LCD

Affichage initial

- On réalise l'affichage initial avec le texte « Hello world !»

```
void setup() {  
  Wire.begin(); // initialisation de la librairie I2C  
  lcd.init(); // initialisation de l'afficheur  
  
  lcd.print("hello, world!"); // affiche message  
}
```

Fonction **loop()**

- Ensuite on se positionne en colonne 1, ligne 2
- et on affiche les secondes :

```
void loop() {  
    lcd.setCursor(0, 1); // positionne le curseur en colonne 1, ligne 2  
    lcd.print(millis()/1000); // affiche les secondes  
}
```

Fonctionnement du programme

- Une fois la carte Arduino programmée, l'afficheur affiche le message programmé :



Voilà, vous savez utiliser un afficheur LCD I2C ! Je ne m'étendrai pas davantage ici, le but étant de simplement montrer le principe.
Noter qu'en dehors de l'initialisation I2C, cet afficheur LCD fonctionne par ailleurs comme avec la librairie LiquidCrystal native.
Pour plus de détails et d'exemples, voir les tutos consacrés à l'utilisation d'un afficheur alpha-numérique qui seront facilement transposables.

Remarque

**Il n'est pas possible dans le cadre de ce tuto d'aller au-delà de ces exemples pour montrer le principe de la communication I2C.
Les fonctionnalités des shields I2C dédiés (RTC, etc...) seront présentés en détail dans des tutos spécifiques.**

24. Rappel : Langage Arduino : les fonctions de la librairie **LiquidCrystal** pour le contrôle des afficheurs LCD standards

Je vous redonne ici la liste des fonctions disponibles pour la gestion d'un afficheur LCD avec la librairie standard, utilisable exactement de la même manière avec l'afficheur I2C Deuligne présenté ici.

Les fonctions de la librairie

La librairie dispose de nombreuses fonctions, à savoir :

- Fonctions d'initialisation : [begin\(\)](#)
- Fonctions d'écriture : [print\(\)](#) | [write\(\)](#)
- Fonctions de gestion de l'écran : [clear\(\)](#) | [display\(\)](#) | [noDisplay\(\)](#) |
- Fonctions de positionnement du curseur : [home\(\)](#) | [clear\(\)](#) | [setCursor\(\)](#)
- Fonctions modifiant l'aspect du curseur : [cursor\(\)](#) | [noCursor\(\)](#) | [blink\(\)](#) | [noBlink\(\)](#)
- Fonctions de contrôle du comportement du curseur : [autoscroll\(\)](#) | [noAutoscroll\(\)](#) | [leftToRight\(\)](#) | [rightToLeft\(\)](#)
- Fonctions d'effets visuels : [scrollDisplayLeft\(\)](#) | [scrollDisplayRight\(\)](#)
- Fonction de création de caractère personnalisé : [createChar\(\)](#)



Pour le détail complet des fonctions de la librairie, voir :

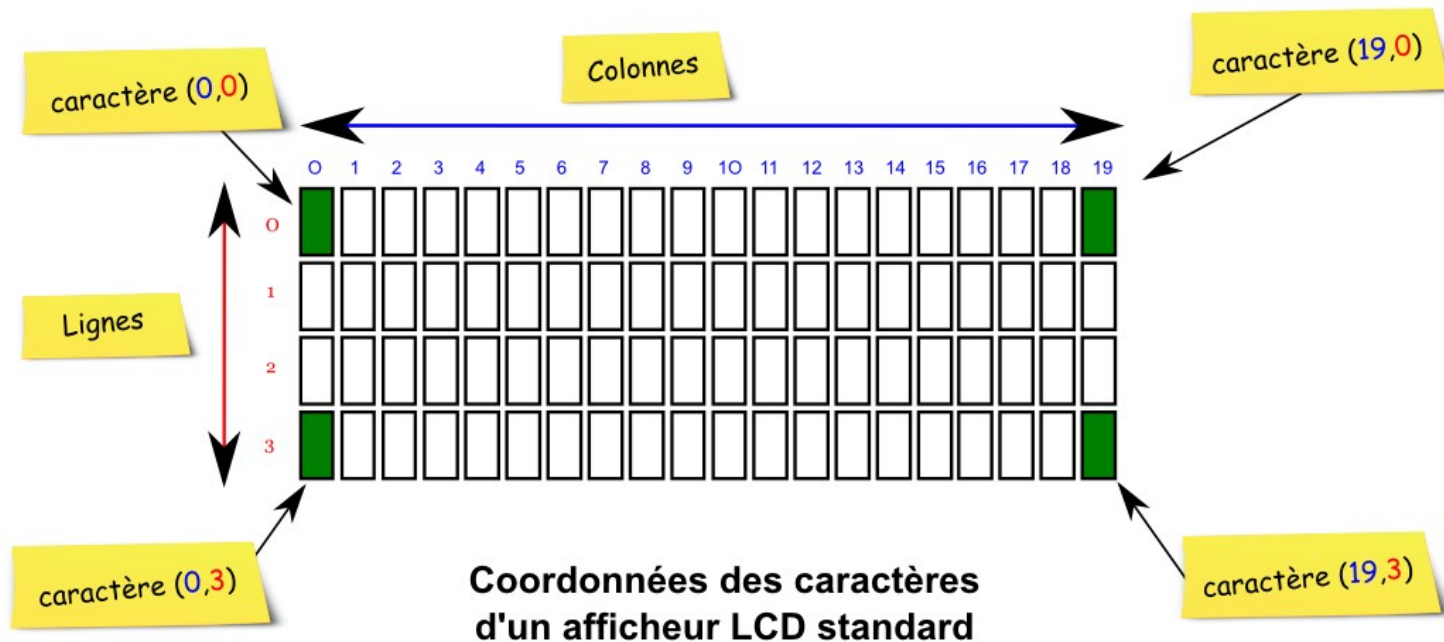
http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieLCD

25. Rappel : Le principe de positionnement sur l'afficheur LCD standard

- Par la suite, nous allons voir les fonctions permettant de se positionner à un emplacement voulu sur l'afficheur. Ceci est rendu possible grâce à une numérotation des caractères sous la forme de coordonnées (colonne,ligne) un peu à la façon (x,y) des points d'un graphique ou encore d'une bataille navale (« colonne A, ligne 2 »).
- Les colonnes sont numérotées de 0 à x-1 pour un afficheur de x colonnes. et les lignes sont numérotées de 0 à y-1 pour un afficheur de y lignes. Par exemple, dans le cas d'un afficheur de 20 colonnes x 4 lignes, les colonnes seront numérotées de 0 à 19 et les lignes de 0 à 3.

Le point important : la première ligne et la première colonne ont le numéro 0

- Voici le schéma résumant ceci :



A RETENIR :

L'instruction `clear()` positionne le curseur en (0,0) c'est à dire dans le coin supérieur gauche de l'afficheur (1er caractère de la 1ère ligne)
Lorsque l'on affiche un caractère avec l'instruction `print()`, ceci a pour effet de décaler la position du curseur d'un cran vers la droite

26. Les éléments du langage Arduino étudiés dans cet atelier

Fonctions de la librairie Wire

Fonctions d'initialisation

- `begin()` : initialise communication avec Arduino "maître"
- `begin(adresse)` : initialise communication avec Arduino "esclave"

Fonctions "mode maître"

- `requestFrom(adresse, quantite)` : demande de données à un esclave
- `beginTransmission(adresse)` : débute communication avec un esclave (ouvre stockage données à envoyer avec `write`)
- `endTransmission()` : envoi des données vers esclave
- `write()` : écrit les données à envoyer vers esclave
- `available()` : test si données disponibles en provenance esclave (cf `requestFrom`)
- `read()` : lit les données en provenance de l'esclave

Fonctions "mode esclave"

- `write()` : envoie les données vers le maître après requête
- `available()` : test si données disponibles en provenance du maître (cf `onReceive`)
- `read()` : lit données en provenance maître
- `onReceive(fonction)` : définit la fonction à appeler sur réception de données en provenance du maître
- `onRequest(fonction)` : définit la fonction à appeler sur requête du maître

Fonctions de la librairie **LiquidCrystal**

Dans cet atelier, les fonctions de la librairie **LiquidCrystal** ont été abordées :

- Fonctions d'initialisation : [`begin\(\)`](#)
- Fonctions d'écriture : [`print\(\)`](#) | [`write\(\)`](#)
- Fonctions de gestion de l'écran : [`clear\(\)`](#) | [`display\(\)`](#) | [`noDisplay\(\)`](#) |
- Fonctions de positionnement du curseur : [`home\(\)`](#) | [`clear\(\)`](#) | [`setCursor\(\)`](#)
- Fonctions modifiant l'aspect du curseur : [`cursor\(\)`](#) | [`noCursor\(\)`](#) | [`blink\(\)`](#) | [`noBlink\(\)`](#)
- Fonctions de contrôle du comportement du curseur : [`autoscroll\(\)`](#) | [`noAutoscroll\(\)`](#) | [`leftToRight\(\)`](#) | [`rightToLeft\(\)`](#)
- Fonctions d'effets visuels : [`scrollDisplayLeft\(\)`](#) | [`scrollDisplayRight\(\)`](#)
- Fonction de création de caractère personnalisé : [`createChar\(\)`](#)

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

27. A présent, vous devriez être capable :

- de comprendre les principes de la communication I2C disponible avec Arduino,
- de savoir utiliser la librairie Wire du langage Arduino
- d'utiliser quelques composants I2C et d'en comprendre le principe d'utilisation,
- d'utiliser des shields utilisant la communication I2C avec Arduino.

Table des matières

Utiliser la communication série I2C (bibliothèque Wire) avec Arduino.

Intro |

Matériel nécessaire pour les ateliers Arduino |

Matériel spécifique nécessaire pour cet atelier |

Technique : Communication I2C : principe général |

Pour info : topologie I2C multi-maîtres / multi-esclaves |

Pour info : le protocole I2C |

Exemple d'enregistrement d'une trame de communication I2C |

Exemple de dispositifs fonctionnant en I2C |

Rappel : Langage Arduino : Introduction aux bibliothèques |

Langage Arduino : la bibliothèque Wire pour l'utilisation de la communication série I2C |

Connexions I2C de la carte Arduino |

Truc technique : utiliser un dispositif ou un shield I2C avec une carte Arduino Mega |

Informations techniques importantes : |

Info : Exemple de composant I2C : le Max7313, extenseur I2C PWM. |

Exemple I2C : le DS1307 : circuit intégré « Temps-réel » RTC (Real Time Clock) à communication I2C |

Utiliser un DS1307 « brut » sur plaque d'essai |

Exemple de shield Arduino utilisant I2C : L'étage RTC du shield « Mémoire » de chez Snootlab |

Exemple I2C : Temps-réel avec un DS1307 : la bibliothèque RTCLib |

Exemple I2C : Temps-réel avec un DS1307 : le montage |

Exemple I2C : Temps-réel avec un DS1307 : le programme |

Exemple I2C : Utiliser un afficheur LCD à communication I2C : prérequis |

Exemple I2C : Utiliser un afficheur LCD à communication I2C : le montage |

Exemple I2C : Utiliser un afficheur LCD à communication I2C : le programme |

Rappel : Langage Arduino : les fonctions de la bibliothèque LiquidCrystal pour le contrôle des afficheurs LCD standards |

Rappel : Le principe de positionnement sur l'afficheur LCD standard |

Les éléments du langage Arduino étudiés dans cet atelier |

A présent, vous devriez être capable : |

Bravo !
vous avez terminé cet atelier Arduino !



Prêt pour la suite ? Retrouvez de nombreux autres thèmes d'ateliers Arduino ici :
http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS