

# Utiliser la communication série SPI (librairie SPI) avec Arduino.



## Ateliers Arduino

par X. HINAULT

[www.mon-club-elec.fr](http://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](mailto: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](mailto: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](mailto: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 !**

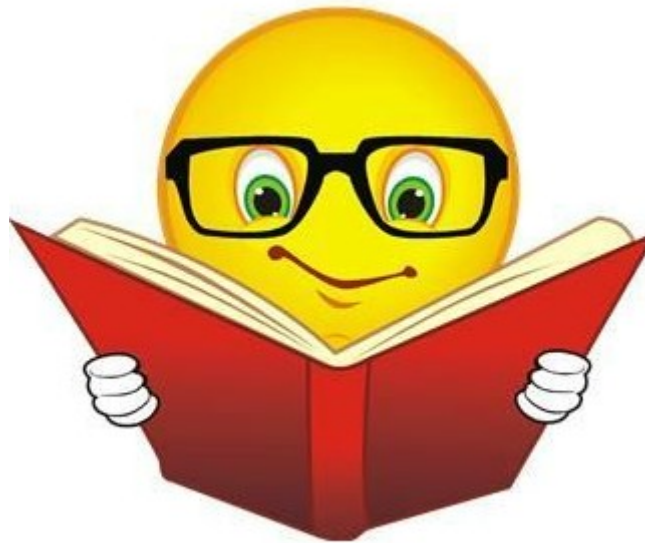
**Lancer également le logiciel Arduino et connecter votre carte Arduino afin de pouvoir tester au fur et à mesure les codes d'exemples !**

## 1. Intro

L'objectif ici est :

- de découvrir et d'apprendre à utiliser la communication SPI (Interface Serie pour Périphérique) disponible avec Arduino,
- de découvrir la librairie SPI du langage Arduino
- à titre d'exemple, vous allez apprendre comment interfacier votre carte Arduino avec une mémoire externe EEPROM série type AT25HP512 qui utilise le protocole de communication série SPI (Interface Série pour Périphérique). Les étapes que nous présenterons ici pour implémenter la communication SPI pourront être modifiées pour être utilisées avec la plupart des autres composants SPI.
- vous découvrirez à titre indicatif quelques shields utilisant la communication SPI

... afin d'être en mesure d'utiliser le protocole de communication SPI avec Arduino.



**Prêt ? C'est parti !**

### **Pratique :**

Les codes de cet atelier sont disponibles ici :

<https://github.com/sensor56/ea2148ab698bd34dfdf530d8c34f05fa>

## 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

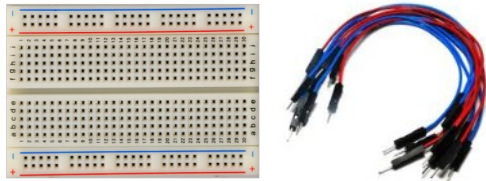


L'espace de développement Arduino associe :

- 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

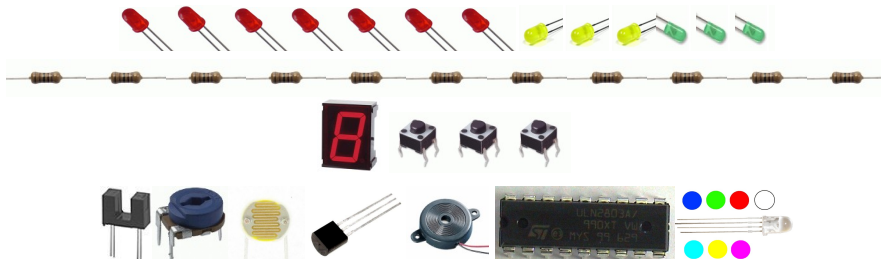


Pour réaliser des montages sans soudure, vous aurez besoin :

- 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](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 potentiellement besoin également :

**D'une Eeprom à communication SPI : par exemple une mémoire Eeprom type AT25256 (ou équiv.)**



- Une Eeprom est une mémoire non volatile électriquement effaçable et qui persiste lors de la mise hors tension. Une Eeprom permet donc de stocker des petites quantités de données.
- L'Eeprom AT25256 est une eeprom contenue dans un boîtier DIL8 (=8 broches)
- Sa capacité est de 256 Kbits soit 32 KOctets (1 octet = 8 bits)

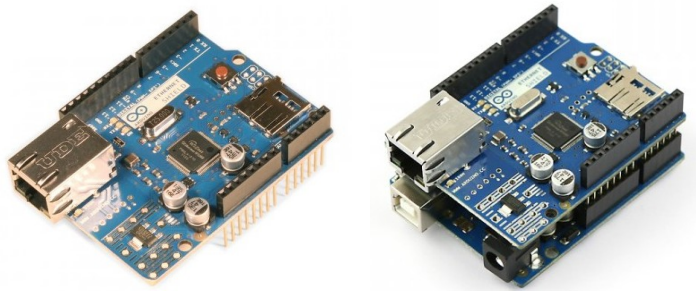
**D'un shield Arduino intégrant un étage utilisant SPI**

- Les cartes d'extension (shields) ou dispositifs utilisant la communication SPI sont nombreux. Notamment :
  - Le shield Arduino Ethernet
  - Les shields utilisant une SD card (shield Ethernet, shield Memoire (Snootlab), ...)
  - Les shields Arduino avec écran graphique couleur TFT, etc...



#### 4. Exemples de shields Arduino utilisant un étage à communication SPI

##### Shield d'exemple utilisé dans ce tuto : une carte d'extension (shield) Ethernet (Arduino)



La carte d'extension (ou shield) ethernet Arduino est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser Arduino sur un réseau ethernet local voire même sur internet.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4 ) pour communiquer avec Arduino.

**Ce shield intègre également un emplacement pour carte mémoire SD** pour des stockage de données ou de pages HTML locales.

Ne pas confondre ce shield avec la carte UNO Ethernet qui est une variante d'une carte UNO avec ethernet intégré.

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

Prix constaté : 33€ environ

##### Carte d'extension (shield) mémoire SD seule (Seeeduno)



La carte d'extension (ou shield) mémoire SD est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser une carte mémoire micro SD, SD et SDHC.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 ) pour communiquer avec Arduino.

disponible chez <http://www.gotronic.fr/> Prix constaté : 12€ environ

##### 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



## 5. Technique : Communication SPI : principe général

### Intro

- L'interface Série pour périphériques (SPI) est un **protocole de communication série synchrone simple** (unidirectionnel pour chaque ligne) utilisé par les microcontrôleurs (tels que la carte Arduino) **pour communiquer avec un ou plusieurs composants** périphériques rapidement sur de courtes distances. Ce protocole peut aussi être utilisé pour des communications entre deux microcontrôleurs.

#### Communication série synchrone ???

En bon français, série ça veut dire que les données seront transmises à la « queue leu-leu » (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.

- 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 : par exemple, sur le shield Ethernet, la communication SPI est utilisée pour communiquer avec la carte SD d'une part et l'étage Ethernet d'autre part.

### Les éléments clés de la communication SPI

- Bon, vous allez voir, c'est très logique... 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 pour chaque sens de communication** comme on l'a dit, il y aura donc logiquement 2 fils de communication, à savoir :
  - un fil du maître vers l'esclave
  - et un fil de l'esclave vers le maître
- 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 :

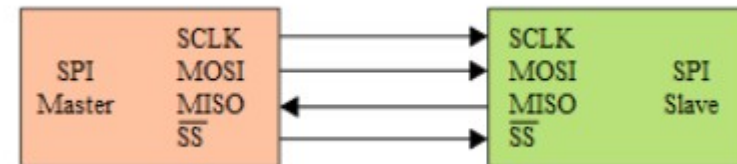
2 éléments qui discutent entre eux : l'esclave et le maître

2 fils de communication : un dans chaque sens

1 fil de synchronisation : le fil d' horloge

### La même chose... en plus technique !

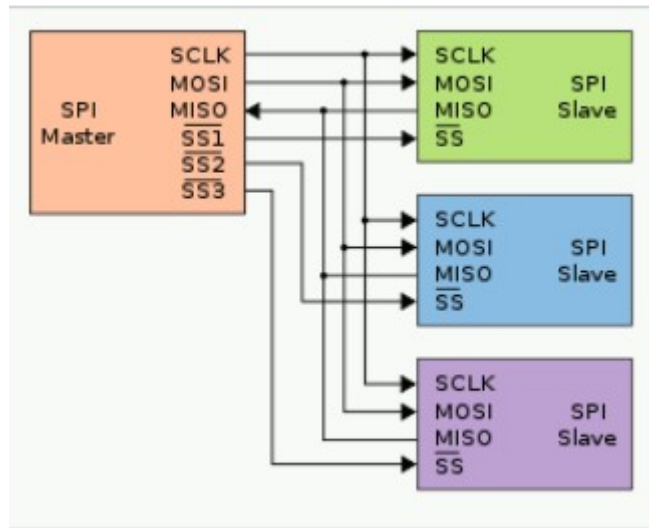
- Dans ce qui suit, on va redire exactement la même chose, mais en utilisant les mots techniques d'usage. Allez c'est parti et ne vous laissez pas impressionné, c'est simple en fait !
- Avec une connexion SPI, il y a toujours un composant maître (habituellement un microcontrôleur, Arduino pour nous) lequel commande-le ou les composants périphériques.
- Typiquement, il y a **3 lignes communes** à tous les composants et qui les relient entre eux :
  - la ligne Maître sortant - Esclave entrant ( ou **MOSI** pour "Master Out Slave In" en anglais) : c'est la ligne utilisée pour envoyer des données du maître vers le(s) périphérique(s).
  - la ligne Maître entrant - Esclave sortant (ou **MISO** pour "Master In Slave Out" en anglais) : c'est la ligne utilisée pour envoyer des données depuis le(s) périphérique(s) vers le maître,
  - la ligne du signal d'horloge ( ou **SCLK** pour Serial Clock en anglais) : cette ligne transmet les impulsions de signal d'horloge générées par le composant maître et qui synchronisent la transmission des données.



- A ces 3 lignes communes, s'ajoutent **une ligne de sélection** par périphérique :
  - la ligne de sélection de l'esclave (ou **/SS** pour "Slave Select" en anglais) : chaque composant périphérique dispose d'une broche de sélection (active sur le niveau BAS) que le composant maître peut utiliser pour activer ou désactiver le(s) composant(s) périphérique(s) voulu(s). **Quand cette broche est mise au niveau HAUT, le composant ignore ce qui se passe sur le bus SPI.**

## 6. Technique : Communication SPI : principe d'utilisation de plusieurs modules

- Comme on vient de le dire, la communication SPI utilise :
  - 3 fils (ou lignes) communes :
    - MOSI : du maître vers l'esclave
    - MISO : de l'esclave vers le maître
    - SCLK : une broche d'horloge qui synchronise la communication
  - 1 fils de sélection pour chaque esclave
- la ligne de sélection de l'esclave (ou /**SS** pour "Slave Select" en anglais) : chaque composant périphérique dispose d'une broche de sélection (active sur le niveau BAS) que le composant maître peut utiliser pour activer ou désactiver le(s) composant(s) périphérique(s) voulu(s). **Quand cette broche est mise au niveau HAUT, le composant ignore ce qui se passe sur le bus SPI.**
- Ceci permet d'avoir plusieurs composants périphériques qui se partagent les 3 lignes de communications MISO, MOSI et SCLK :
  - le maître pourra être connecté à plusieurs « esclaves » :
    - les lignes MISO, MOSI et SCLK seront communes
    - mais chaque esclave sera relié à une seule broche de sélection du maître.
  - Ainsi, en mettant une seule des broches de sélection du maître au niveau BAS à la fois, le maître peut ainsi définir avec quel « esclave », il « parle »



### Ce principe fait la grande force de la communication SPI :

en effet, il sera ainsi possible d'utiliser plusieurs dispositifs SPI avec Arduino et de communiquer avec en utilisant les mêmes broches de communication pour tous les modules (MOSI, MISO, SCLK) et une broche de sélection pour chaque module.

Dans le cas de 3 dispositifs, on utilise ainsi 3 communes + 3 sélection soit 6 broches Arduino au lieu de  $3 \times 4 = 12$  broches soit 50 % de broches en moins !

## 7. Technique : Principe de mise en oeuvre de la communication SPI

### Intro

- Je vous donne ici tous les détails du fonctionnement de la communication SPI pour pouvoir utiliser directement un composant SPI : **sachez qu'il est possible le plus souvent de se simplifier la vie en utilisant une librairie dédiée à chaque composant, comme nous le verrons ensuite.**
- Le standard SPI est libre et chaque composant l'implémente de façon légèrement différente. Cela signifie que vous devrez donner une attention spéciale à la lecture de la fiche technique (datasheet) du composant utilisé quand vous écrirez votre code.

### Les questions à se poser

- Pour écrire un programme pour un nouveau composant SPI, il faut prendre en compte plusieurs éléments :
  - Les **données série** sont-elles envoyées avec le bit de poids fort (ou MSB pour "Most Significant Bit" en anglais) ou avec le bit de poids faible (ou LSB pour "Least Significant Bit" en anglais) en premier ? Ceci est contrôlé par la fonction [SPI.setBitOrder\(\)](#).

#### Pour comprendre :

On a dit que les données, c'est-à-dire les 0 et les 1 composants les octets, seraient envoyés en mode série, autrement dit à la « queue leuleu ».

Par exemple le nombre 240 en binaire sera **11110000**.

Il y a 2 façons de l'envoyer :

soit le bit de **poids faible** en premier :== **11110000** ==>

soit le bit de **poids fort** en premier : == **00001111** ==>

- Le **signal d'horloge** est-il inactif sur le niveau HAUT ou le niveau BAS ?
- Les **données** sont-elles prises en compte sur le **front montant (FM)** ou le **front descendant (FD)** de l'impulsion d'horloge ? Ceci et le niveau actif de l'horloge sont contrôlés par la fonction [SPI.setDataMode\(\)](#)
- A quelle vitesse le bus SPI fonctionne-t-il ? Ceci est contrôlé par la fonction [SPI.setClockDivider\(\)](#)

### Les modes de transmission possibles

- De façon générale, il y a 3 modes de transmission. Ces modes définissent :
  - la façon dont les données sont envoyées en entrée et en sortie sur le front montant ou descendant du signal d'horloge (appelé également la **phase d'horloge**),
  - si le signal d'horloge est inactif sur le niveau HAUT ou BAS (appelé également la **polarité d'horloge**).
- Ces trois modes combinent la polarité et la phase d'horloge. La fonction [SPI.setDataMode\(\)](#) vous laisse définir mode de fonctionnement en fonction de la polarité et de la phase de l'horloge, en accord avec ce tableau :

Mode	Clock Polarity (CPOL)	Clock Phase (CPHA)
0	0	0
1	0	1
2	1	0
3	1	1

### Et pour finir...

- Une fois que vos paramètres SPI sont configurés correctement, il ne vous reste plus qu'à déterminer quels registres de votre composant contrôlent quelles fonctions, et vous alors opérationnel. Ceci sera expliqué dans la fiche technique (datasheet) de votre composant.



« Euh... tu peux nous la refaire là ! J'ai rien pigé »

Pas de panique... ça va se décanter avec l'exemple à suivre !



## 8. Connexions SPI de la carte Arduino

### Les broches Arduino pour la communication SPI

- Très logiquement, Arduino va disposer des 3 broches communes MOSI, MISO et SCLK ainsi que d'une broche de sélection par défaut appelée /SS
- Sur la carte Arduino UNO, Duemilanove et autres cartes basées sur les ATmega 168 / 328, le bus SPI utilise les broches :
  - 10 : /SS
  - 11 : MOSI
  - 12 : MISO
  - et 13 : SCLK.
- Sur l'Arduino Mega, ce sont les broches :
  - 50 : MISO
  - 51 : MOSI
  - 52 : SCLK
  - et 53 : /SS

### Bon à savoir

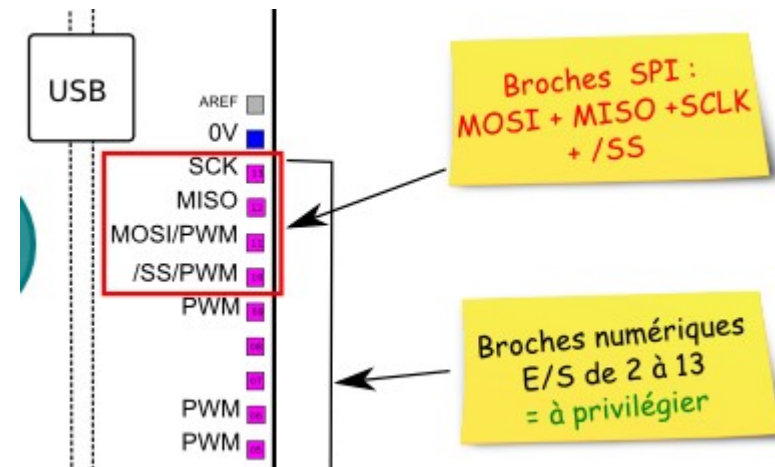
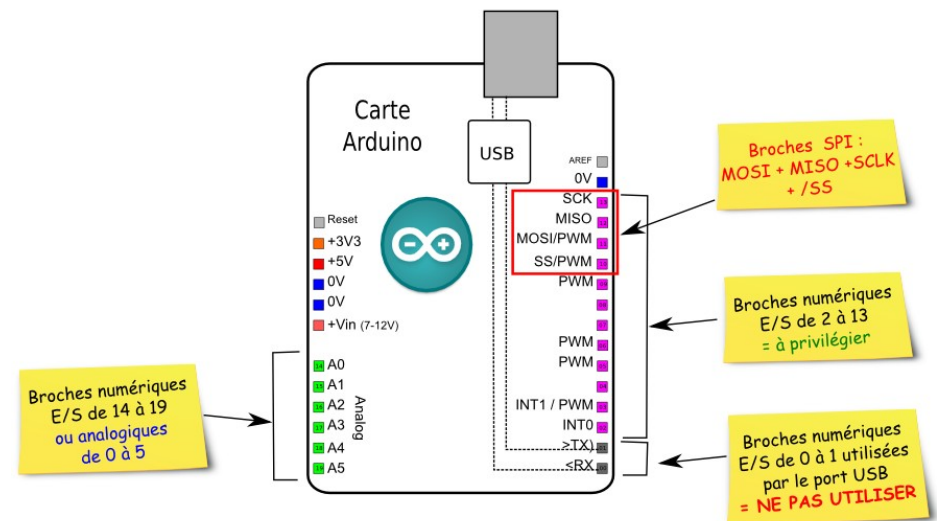
- Sur la carte UNO et la Mega, les broches 1, 3 et 4 du connecteur ICSP sont connectées aux broches MISO, MOSI et SCLK. Ceci permet notamment l'utilisation de modules Arduino indifféremment avec une carte UNO et Mega, la connexion SPI se faisant par les 3 broches du connecteur ICSP connectées au bus SPI. C'est le cas du shield Ethernet notamment.

### Broche de sélection

- Il est possible d'utiliser une autre broche que la broche 10 en tant que broche de sélection de périphérique ( /SS pour Slave Select - active sur niveau BAS).
- Par exemple, le [module Arduino Ethernet](#) utilise la broche 4 pour activer la connexion SPI avec la carte mémoire SD et la broche 10 pour activer la connexion SPI avec l'interface Ethernet.

### Important !

Noter que **même si vous n'utilisez pas la broche /SS, elle doit quand même être configurée en sortie**; sinon, l'interface SPI sera mise en mode "esclave" et rendra la librairie inopérante.

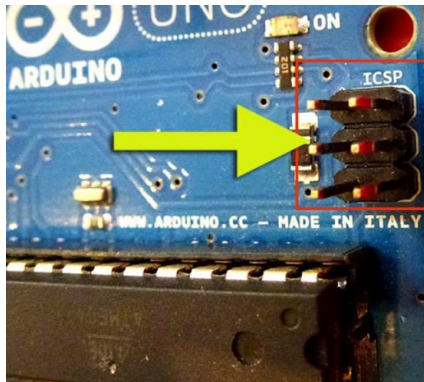


Remarquer que l'utilisation de la communication SPI fait également perdre 2 broches PWM (la 10 et la 11) sur la UNO/Duemilanove.

## 9. Truc technique : utiliser un dispositif SPI avec une carte Arduino Mega

### Pour les dispositifs intégrant un connecteur ICSP femelle

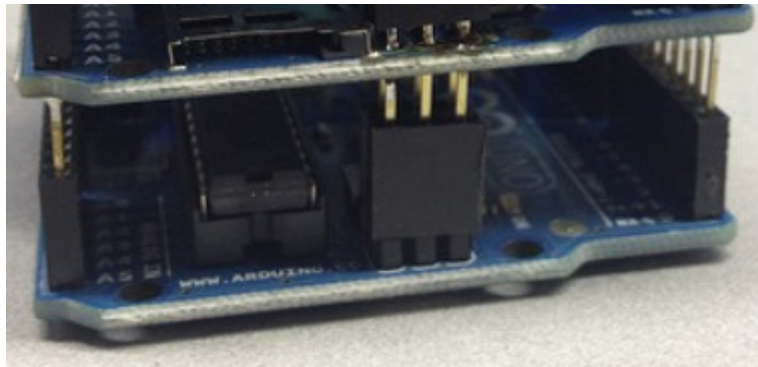
- La plupart des cartes Arduino disposent d'un connecteur droit 6 broches, appelé connecteur ICSP.



- Ce connecteur droit reprend les broches MISO, MOSI et SCK selon :

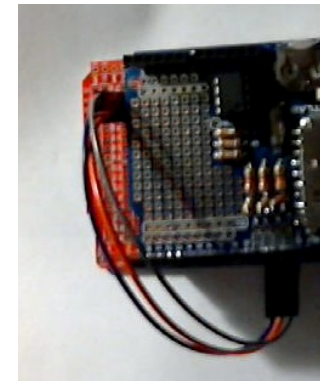


- L'intérêt majeur est de pouvoir utiliser indifféremment sur une carte Arduino UNO ou Mega un dispositif ou un shield possédant un connecteur femelle ICSP, comme c'est le cas par exemple pour le shield Ethernet.



### Pour les dispositifs sans connecteur ICSP femelle

- Dans ce cas, la « combine » consiste à « plier » latéralement les pattes 11,12 et 13 ( MOSI, MISO et SCK ) du shield utilisé à les connecter par des jumpers aux broches 51, 50 et 52 (MOSI,MISO et SCK) de la carte Arduino Mega.
- Et en faisant cela pour le premier shield enfiché sur la carte, on pourra empiler d'autres shields utilisant SPI par dessus !
- Attention les broches MOSI/MISO ont un ordre inverse par rapport à SCK entre la UNO et la Mega (MOSI : 11 vers 51 et MISO : 12 vers 50)
- Le dernier point important sera d'utiliser une broche de sélection pour le dispositif qui peut rester la 10 comme avec Arduino UNO.
- Enfin, il faudra **mettre en sortie la broche 53** de la Mega qui est la broche /SS par défaut de la carte Arduino Mega, pour activer SPI.



Voici la synthèse des broches SPI utilisées pour les cartes Arduino :

Arduino Board	MOSI	MISO	SCK	SS (slave)	SS (master)
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	-
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	-
Leonardo	ICSP-4	ICSP-1	ICSP-3	-	-
Due	ICSP-4	ICSP-1	ICSP-3	-	4, 10, 52

## 10. Rappel : Langage Arduino : Les 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; // déclare 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>

## 11. Langage Arduino : la librairie **SPI** pour l'utilisation de la communication série SPI

### Présentation

- La librairie **SPI** est une librairie standard Arduino qui propose toutes les fonctions utiles pour mettre en œuvre simplement une communication série simplex SPI.

### Inclusion

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

```
#include <SPI.h> // inclut la librairie SPI
```

### 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

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

### Les fonctions de la librairie

Les instructions disponibles sont très logiquement :

- begin()** : Initialise le bus SPI
- end()** : Désactive le bus SPI
- setBitOrder()** : Configure l'ordre des bits transmis en sortie ou en entrée sur le bus SPI
- setClockDivider()** : Configure le diviseur d'horloge du bus SPI par rapport à l'horloge système.
- setDataMode()** : Configure le mode de transmission des données sur le bus SPI
- transfer()** : Transfère un octet sur le bus SPI, aussi bien en envoi qu'en réception.

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

[http://www.mon-club-elec.fr/pmwiki\\_reference\\_arduino/pmwiki.php?n=Main.LibrairieSPI](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.LibrairieSPI)

### Principe d'utilisation

La structure type d'un programme utilisant la communication SPI sera logiquement :

#### Au niveau de l'entête déclarative

- Inclusion de la librairie **SPI**
- Déclaration de la broche de sélection utilisée.
- Définition éventuelle de constantes correspondant aux octets d'instructions reconnues par le dispositif SPI utilisé.

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

- Initialisation de la communication SPI avec l'instruction **SPI.begin()**
- Paramétrage du mode de fonctionnement utilisé pour la communication SPI (à adapter au dispositif utilisé) :
  - configuration de l'ordre de transmission des bits :
    - SPI.setBitOrder(LSBFIRST)** si bit de poids faible en premier,
    - SPI.setBitOrder(MSBFIRST)** si bit de poids fort en premier.
  - configuration du diviseur d'horloge (2 à 128) pour le bus SPI :
    - SPI.setClockDivider(SPI\_CLOCK\_DIV4)** par exemple
  - configuration du mode transmission (parmi 1 des 4 modes) :
    - SPI.setDataMode(SPI\_MODE1)** par exemple
- Mettre la broche de sélection en sortie (**et toujours laisser en sortie la broche 10 (/SS) même si pas utilisée ++**)

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

- Envoi à la demande des octets vers le dispositif SPI avec la fonction :
  - SPI.transfer(valeurOctet)**

### Important

Bien comprendre que le protocole de communication SPI est indépendant des commandes et du fonctionnement propre du dispositif SPI avec lequel Arduino communiquera. Les règles de communication sont expliquées et détaillées dans la fiche technique du dispositif en question.



## 12. Pour info : Arduino : les rouages internes de la communication SPI

Je donne les infos qui suivent uniquement à but informatif, pour les curieux... Vous pouvez passer à la suite cela ne nous intéresse pas : vous n'en n'aurez pas absolument besoin pour la suite. Cela vous présente cependant les rouages utilisés par la librairie SPI.



- Un microcontrôleur, en interne, c'est un peu comme un cockpit d'avion :
  - il y a pleins d'interrupteurs On/Off qui permettent d'activer/désactiver des fonctions : les bits d'activation.
  - il y a plein de voyants qui signalent tel ou tel événement : les bits de drapeau.
- Tous les paramétrages de la communication SPI sont déterminées pour les cartes Arduino par le **registre de contrôle SPI (SPCR)**. Un registre est juste un octet de la mémoire du microcontrôleur qui peut être lu ou écrit. Les registres ont généralement trois fonctions : contrôle, donnée et état.
- Les **registres de contrôle** fixent les paramètres de contrôle pour toute une variété de fonctionnalités du microcontrôleur. Habituellement, chaque bit dans un registre de contrôle agit sur un paramètre précis, tel que la vitesse ou la polarité de l'horloge.
- Les **registres de données** contiennent simplement des octets. Par exemple, le registre de donnée SPI (SPDR) contient l'octet qui doit être envoyé sur la ligne MOSI et les données qui viennent d'être reçues par la ligne MISO.
- Les **registres d'état** changent leur état en fonction de conditions variées liées au fonctionnement du microcontrôleur. Par exemple, le 7ème bit du registre d'état (SPSR) est mis à 1 quand une valeur est émise ou reçu par le module SPI du micro-contrôleur.
- Le registre de contrôle SPI (SPCR) a 8 bits, chaque bit contrôlant un paramètre particulier du module SPI.

### Le registre de contrôle SPCR

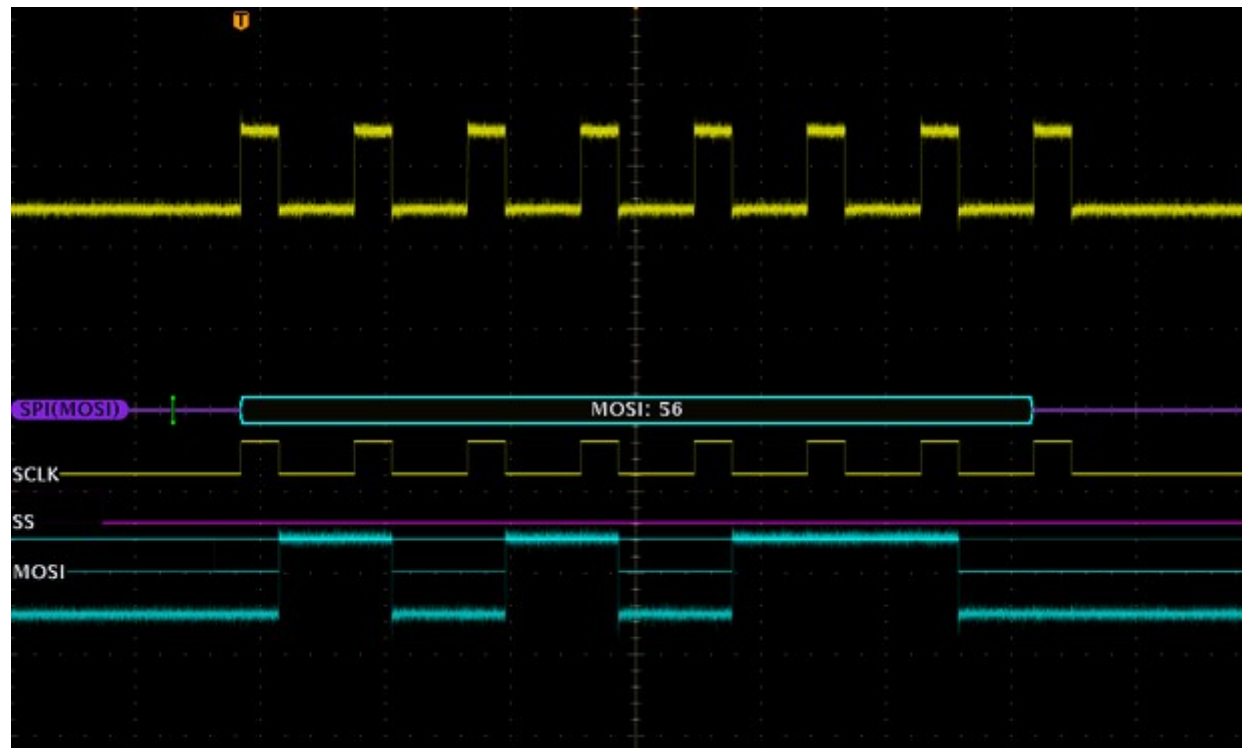
Bit	7	6	5	4	3	2	1	0
Nom	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
avec :								

- SPIE (Enable Interrupt SPI bit) : Activation de l'interruption SPI si égal à 1
- SPE (Enable SPI) : Active le module SPI si égal à 1
- DORD : Envoie les données en commençant par le bit de poids faible si égal à 1, par le bit de poids fort si égal à 0.
- MSTR : Configure l'Arduino en mode MAÎTRE si égal à 1, en mode ESCLAVE si égal à 0
- CPOL : Configure l'impulsion d'horloge inactive au niveau HAUT si égal à 1, inactive au niveau BAS si égal à 0
- CPHA : Valide les données sur le front descendant si vaut 1, sur le front montant si vaut 0
- SPR1 et SPR0 : Configure la vitesse de communication : 00 est le plus rapide ( $F_{osc}/4 = 4\text{MHz}$ ), 11 est le plus lent (250KHz)

La librairie SPI fournie avec le langage Arduino simplifie grandement la gestion de la communication SPI, mais je vous donne ces informations techniques pour la communication SPI n'aie plus aucun secret pour vous.

### 13. Pour info : Ce qui se passe sur le bus SPI pendant un échange « Maître - Esclave »

- Voici une capture d'écran d'un enregistrement par oscilloscope numérique de ce qui se passe sur les lignes MISO, MOSI, SCLK et /SS lors d'un échange SPI :



Il est évident ici que la communication SPI repose sur une complexité sous-jacente difficile à maîtriser si l'on débute avec Arduino...  
Cette même complexité, une fois prise en main, ouvre la voie à de nombreuses possibilités passionnantes !



## 14. Exemple de dispositifs fonctionnant en SPI

- Les cartes d'extension (shields) ou dispositifs utilisant la communication SPI sont nombreux. Notamment :
  - Le shield Arduino Ethernet
  - Les shields utilisant une SD card (shield Ethernet, shield Memoire (Snootlab), ...)
  - Les shields Arduino avec écran graphique couleur TFT
  - etc...



Comme vous le constatez ici, la communication SPI vous ouvre un univers de possibilités nouvelles passionnantes et qui sont mises en œuvre de façon relativement simple, grâce à la communication SPI comme nous allons le voir.



### Remarque générale

**Je tiens à redire ici qu'à mon sens, ce type de capteurs et dispositifs SPI 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 très 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...

Je conseille également de ne pas s'initier en utilisant d'emblée des dispositifs nécessitant de bonnes connaissances sous-jacentes (par exemple, l'utilisation d'un shield Ethernet impose de connaître de notions en réseau ethernet, en Http, HTML, etc...)

#### Utiliser des dispositifs SPI ?

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 SPI est vraiment passionnante et ouvre des possibilités nouvelles  
**à réserver cependant aux utilisateurs ayant déjà une bonne expérience avec Arduino,**  
**ce qui sera votre cas très prochainement en utilisant les nombreux tutos que je vous propose !**

## 15. Informations techniques importantes à retenir en pratique

- **Noter que même si vous n'utilisez pas la broche /SS, elle doit quand même être configurée en sortie; sinon, l'interface SPI sera mise en mode "esclave" et rendra la librairie inopérante.** En pratique, ça veut dire **mettre OBLIGATOIREMENT en sortie la broche 10 sur la UNO, la broche 53 sur la Mega.**
- Il est possible d'utiliser une autre broche que la broche 10 en tant que broche de sélection de périphérique ( /SS pour Slave Select - active sur niveau BAS). Par exemple, le [module Arduino Ethernet](#) utilise la broche 4 pour activer la connexion SPI avec la carte mémoire SD et la broche 10 pour activer la connexion SPI avec l'interface Ethernet.
- Ne désactivez pas les interruptions dans les programmes utilisant la communication SPI



**« Merci pour toutes ces infos... mais moi, je veux coder, passer à l'action quoi !!!! »**

Et bien justement, nous y sommes... on va enfin écrire un code utilisant la communication SPI.

Je vous propose de faire ça en 2 temps :

Tout d'abord écrire un code « brut » utilisant directement la communication SPI, ce qui en pratique n'est pas la façon de faire la plus courante.

Puis, des codes utilisant une librairie elle-même basée sur la librairie SPI, ce qui est la situation la plus fréquente, et heureusement d'ailleurs !

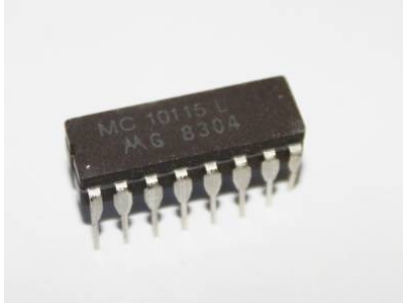
Prêt ? C'est parti... !

## 16. Rappel : Info technique : les circuits intégrés en boîtier DIL

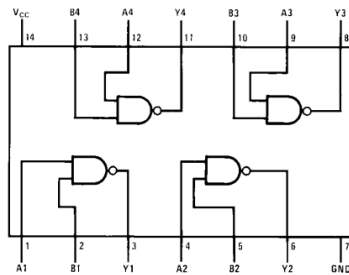
### Présentation des circuits intégrés DIL...

En fait, il s'agit d'un format de boîtier de circuit intégré courant :

- petit boîtier noir en plastique rigide
- présentant toute une série de broches métalliques latérales
- avec un numéro dessus correspondant à la référence du circuit



Un circuit intégré est un circuit électronique miniaturisé et ayant une fonction particulière fixe : un compteur, une bascule JK, portes logiques, etc...



Exemple de schéma fonctionnel – 74LS00 – 4 portes NAND

Un circuit intégré dispose de plusieurs broches :

- Chaque broche du circuit intégré DIL va avoir une fonction précise qui est décrite dans une fiche technique appelée « datasheet » en anglais.
- Dans le cas d'un circuit intégré numérique, les broches métalliques seront des entrées ou sorties numériques.
- Un circuit intégré type dispose également, comme tout dispositif électrique de 2 broches d'alimentation : le 0V et le +5V classiquement.

Les circuits intégrés numériques existent en plusieurs familles :

- circuits analogiques, numériques, mixtes,
- chaque famille est dénommée avec des lettres et des nombres
- l'une d'entre elle est courante : les 74LSxx où xx est un nombre.
- il en existent pleins d'autres, les lettres utilisées étant souvent associées à un fabricant et le nombre à une fonction

74LS00	4 portes NAND à 2 entrées	DIL14
74LS02	4 portes NOR à 2 entrées	DIL14
74LS03	4 NAND à 2 entrées col. ouv.	DIL14
74LS04	Sextuple inverseur	DIL14
SN7406	Sextuple inverseur buffer	DIL14
74LS06	Sextuple inverseur buffer	DIL14
74LS07	Sextuple buffer	DIL14
74LS08	4 portes AND à 2 entrées	DIL14
74LS09	4 AND à 2 entrées col. ouv.	DIL14
74LS09D-CMS	4 AND à 2 entrées col. ouv.	SO14
74LS12	3 NAND à 3 entrées col. ouv.	DIL14
74LS14	Sextuple inverseur trigger	DIL14
74LS15	3 AND à 3 entrées col. ouv.	DIL14

Exemple de nom de circuit et leur fonction



Position type des broches 0V et +5V d'un CI logique.

Noter le sens du circuit indiqué par une marque sur le boîtier.

Avec Arduino, on aura très peu besoin de circuits intégrés externes, la plupart des fonctions pouvant être réalisées par programmation. Un programme élaboré peut être l'équivalent de plusieurs dizaines de circuits intégrés associés !

### 17. Technique : Composant : Un exemple de composant SPI : une mémoire Eeprom type AT25256

## Fonction

- Une Eeprom est une mémoire non volatile électriquement effaçable et qui persiste lors de la mise hors tension. Une Eeprom permet donc de stocker des petites quantités de données.

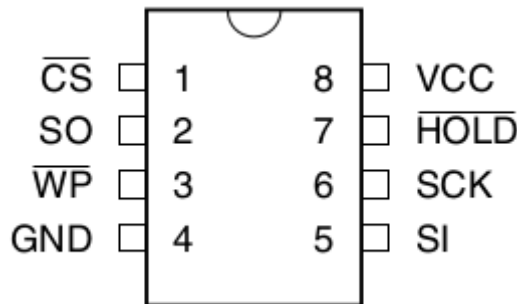
### Description

- L'Eeprom AT25256 est une eeprom contenue dans un boîtier DIL8 (=8 broches)
- Sa capacité est de 256 Kbits soit 32 KOctets (1 octet = 8 bits)

## Brochage

Le brochage est logique :

- 2 broches d'alimentation : OV (broche 9) et V+ (broche 10).
- les 3 broches commune de communication SPI : MOSI, MISO et SCLK
- la broche de sélection /CS
- et 2 broches d'usage spécifique :
  - /HOLD : broche de désactivation SPI
  - /WP : broche de protection en écriture



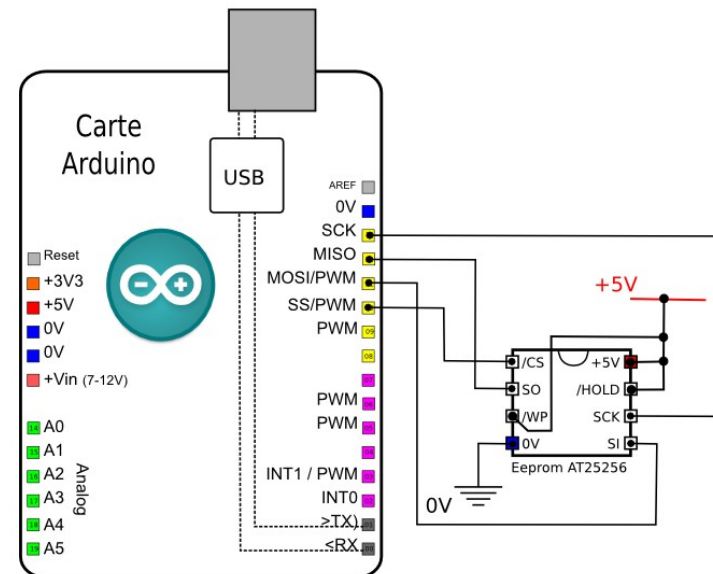
Je vous présente cette Eeprom à titre d'exemple,  
sachant qu'il est en existe de nombreux modèles différents :  
les principes présentés ici seront transposables au besoin.

## Principe de fonctionnement

- La logique interne de cette Eeprom SPI « comprend » les instructions suivantes :
  - WREN (110) : Activation écriture
  - WRDI (100) : Désactivation écriture
  - RDSR (101) : Lecture registre interne de statut
  - WRSR (001) : Ecriture registre interne de statut
  - READ (011) : Lecture donnée de l'Eeprom
  - WRITE(010) : Ecriture donnée dans l'Eeprom
- Les données sont stockées dans des « cases » ayant un numéro appelé « adresse » : de 0 à 32768 soit en hexadécimal de 0x0000 à 0x7FFF
- Typiquement pour écrire une donnée, il faudra envoyer sur le bus SPI :
  - WREN puis WRITE puis Octet Adresse puis les Octets de données

### Circuit de fonctionnement type

- Le circuit minimum de fonctionnement avec Arduino est le suivant :



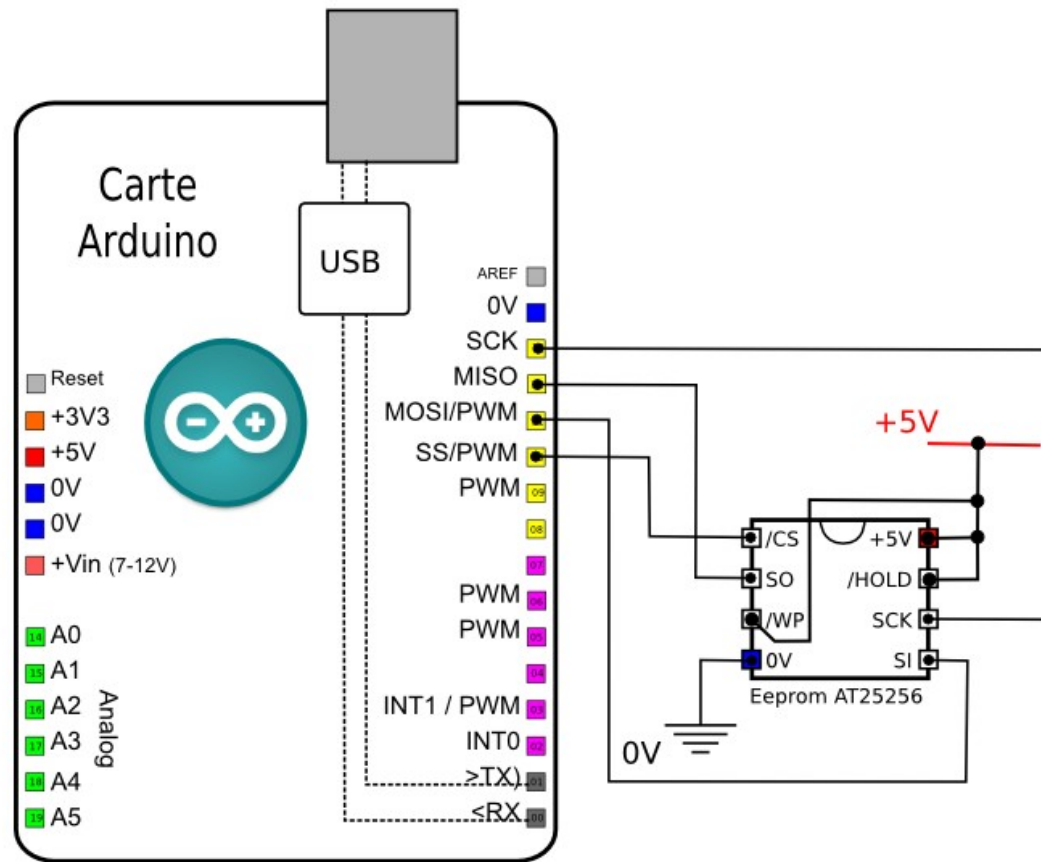
## 18. Exemple d'utilisation SPI : Communiquer directement avec une mémoire Eeprom SPI : le montage

Une fois n'est pas coutume : la reproductibilité de cet exemple est dépendant de la disponibilité de l'Eeprom, pas forcément facile à trouver (dispo chez Farnell )

Vous pourrez facilement transposer cet exemple à toute Eeprom comparable le cas échéant, le but ici étant surtout de montrer le principe.

Si vous n'en disposez pas, vous pouvez vous contenter de lire et passer aux exemples suivants qui correspondent davantage à ce que l'on fait en pratique.

- Nous allons tout simplement connecter :
  - les broches communes MISO, MOSI et SCLK d'Arduino et du composant SPI entre-elles
  - la broche de sélection /SS d'Arduino à la broche /CS du composant SPI
  - les broches spécifiques (/HOLD et /WP dans notre cas) seront laissées connectées en mode inactif, soit +5V ici.





## 19. Exemple d'utilisation SPI : Communiquer directement avec une mémoire Eeprom SPI : le programme



### Remarque :

Je vous propose ce programme ici uniquement à but d'information pour vous montrer comment mettre en place une communication « directe » avec un composant SPI. En pratique, le plus souvent, une librairie est fournie avec un dispositif SPI qui gère toute la communication « directe ».

En un mot, lisez et regardez ce code simplement pour comprendre le principe, avant de passer à la suite.

### Ce qu'on va faire ici...

- Dans ce programme, nous allons tout simplement tester l'écriture et la lecture de données dans une Eeprom SPI en utilisant la communication SPI.

### Le principe

- Comme dit précédemment, l'Eeprom SPI utilisée dispose d'une logique interne qui « comprend » les instructions suivantes :
  - WREN (011) : Activation écriture
  - WRDI (100) : Désactivation écriture
  - RDSR (101) : Lecture registre interne de statut
  - WRSR (001) : Ecriture registre interne de statut
  - READ (011) : Lecture donnée de l'Eeprom
  - WRITE(010) : Ecriture donnée dans l'Eeprom
- Typiquement pour écrire une donnée, il faudra envoyer sur le bus SPI : WREN puis WRITE puis Octet Adresse puis les Octets de données

## Entête déclarative

### Inclusion des bibliothèques utiles

- On commence par inclure les bibliothèques
  - ici la bibliothèque Arduino **SPI** qui implémente la communication SPI

### Déclaration de constantes utiles

- on déclare des constantes binaires correspondant chacune aux instructions reconnues par l'Eeprom SPI (voir doc de l'Eeprom pour les détails)

### Déclaration des broches utilisées

- On déclare ici la broche utilisée pour sélectionner l'Eeprom SPI

### Variables utiles

- On déclare plusieurs variables utiles, notamment pour la gestion de l'adresse

```
// --- Inclusion des bibliothèques ---
#include <SPI.h> // Bibliothèque pour communication SPI

// --- Déclaration des constantes utiles ---

// --- les codes des instructions de l'eeprom SPI
const int WREN=B110; // code instruction Write Enable
const int WRDI=B100; // code instruction Write Reset
const int RDSR=B101; // code instruction Read Status Register
const int WRSR=B001; // code instruction Write Status Register
const int READ=B011; // code instruction Read Data from memory
const int WRITE=B010; // code instruction Write Data to memory

// --- Déclaration des constantes des broches E/S numériques ---

//const int DATAOUT=11; //MOSI
//const int DATAIN=12; //MISO
//const int SPICLOCK=13; //sck
const int chipSelectPin=10; //ss (Slave Select)

// --- Déclaration des constantes des broches analogiques ---

// --- Déclaration des variables globales ---
unsigned int adresse=2000; // adresse 16 bits
byte adresseMSB, adresseLSB; // variables pour adresse
byte data=0;
long memoMillis=0; // pour mémoriser valeur renvoyée par millis
```

## Fonction **setup()**

### Initialisation série

- On commence par initialiser la connexion série qui sera utilisée pour afficher des messages

### Configuration broche utilisée

- On initialise la communication SPI en mode « maître » avec l'instruction **SPI.begin()**
- Puis on configure le fonctionnement de la communication SPI :
  - configuration de l'ordre de transmission des bits :
    - **SPI.setBitOrder(LSBFIRST)** si bit de poids faible en premier,
    - **SPI.setBitOrder(MSBFIRST)** si bit de poids fort en premier.
  - configuration du diviseur d'horloge (2 à 128) pour le bus SPI :
    - **SPI.setClockDivider(SPI\_CLOCK\_DIV4)** ici
  - configuration du mode transmission (parmi 1 des 4 modes) :
    - **SPI.setDataMode(SPI\_MODE0)** ici

### Configuration broche utilisée

- On termine par mettre la broche de sélection en sortie.

```
void setup() { // debut de la fonction setup()

Serial.begin(115200); // initialise connexion série à 115200 bauds
// IMPORTANT : régler le terminal côté PC avec la même valeur de transmission

// initialisation de la librairie SPI
SPI.begin(); // Arduino configuré en maitre

//----- sens bits ----
SPI.setBitOrder(MSBFIRST); // fixe le sens de la communication

//----- Configuration Mode (Impulsion horloge inactive et Front validation données
// Mode      Horloge inactive   Front validation donnée
// SPI_MODE0  0                  0
// SPI_MODE1  0                  1
// SPI_MODE2  1                  0
// SPI_MODE3  1                  1
SPI.setDataMode(SPI_MODE0); // l'eprom AT25256 fonctionne en mode 0

//----- configuration de l'horloge ----
SPI.setClockDivider(SPI_CLOCK_DIV4); // le plus rapide

//---- met en sortie la broche de sélection de l'eprom
pinMode(chipSelectPin, OUTPUT);

} // fin de la fonction setup()
```

## Fonction **loop()** : Ecriture d'une donnée dans l'Eeprom

- Avant chaque envoi d'une instruction, il faut sélectionner l'Eeprom (broche CS à LOW) et après l'envoi, remettre la broche à HIGH.
- On commence ainsi par envoyer l'instruction d'activation de l'écriture WREN avec la fonction **transfer()**
- Puis, ensuite, on envoie l'instruction WRITE, suivie de l'adresse sur 2 octets puis de la donnée sur 1 octet, avec la fonction **transfer()**
- on attend enfin la fin de l'écriture en lisant l'état du registre de statut avec la fonction **transfer()** et en testant l'état du bit Ready

```
void loop(){ // debut de la fonction loop()

// activation écriture
digitalWrite(chipSelectPin, LOW); // sélection l'eeprom
SPI.transfer(WREN); // active l'écriture - à faire au début séquence écriture
digitalWrite(chipSelectPin, HIGH); // dé-sélectionne l'eeprom

    delay(10); // pause

//---- message debug
memoMillis=millis(); // mémorise valeur millis()
Serial.print("Millis debut ecriture: "),Serial.println(memoMillis);

// ---- écriture donnée dans l'EEPROM
digitalWrite(chipSelectPin, LOW); // sélection l'eeprom

    SPI.transfer(WRITE); // instruction écriture

    //adresse=70;
    adresseMSB=adresse>>8; // isole les 8 bits forts
    adresseLSB= adresse; // isole les 8 bits faibles

    SPI.transfer(adresseMSB); // envoie MSB adresse en 1er
    SPI.transfer(adresseLSB); // envoie LSB adresse

    data=data+1; // donnée à écrire valeur 0 - 255
    SPI.transfer(data); // écriture donnée eeprom

    digitalWrite(chipSelectPin, HIGH); // dé-sélectionne l'eeprom - obligatoire pour lancer l'écriture

//---- teste la fin du cycle écriture ---
digitalWrite(chipSelectPin, LOW); // sélection l'eeprom

while(bitRead(SPI.transfer(RDSR),0)!=0); // attend que le bit Ready du registre statut passe à 0

    digitalWrite(chipSelectPin, HIGH); // dé-sélectionne l'eeprom
```

## Fonction **loop()** : Lecture de la donnée en Eeprom

- Puis on affiche un message affichant le délai utilisé pour l'écriture
- La lecture de la donnée se fait de façon comparable : on envoie l'instruction READ avec la fonction transfer() suivie de l'adresse à utiliser et d'une nouvelle fonction transfer() en passant une valeur quelconque (obligatoire pour avoir une réponse)
- L'Eeprom est finalement désélectionnée.
- Le cycle recommence après une pause de 1 seconde.

```
//---- message debug
memoMillis=millis(); // mémorise valeur millis()
Serial.print("Millis fin ecriture: "),Serial.println(memoMillis);

    delay(5); // pause pour laisser temps eeprom travailler

//---- lecture donnée eeprom
digitalWrite(chipSelectPin, LOW); // sélection l'eeprom
SPI.transfer(READ); // active la lecture

    //adresse=70;
    adresseMSB=adresse>>8; // isole les 8 bits forts
    adresseLSB= adresse; // isole les 8 bits faibles

    SPI.transfer(adresseMSB); // envoie MSB adresse en 1er
    SPI.transfer(adresseLSB); // envoie LSB adresse

data=0;
data=SPI.transfer(0xFF); // lecture de la donnée...
// pourquoi 0xFF ? en fait la fonction attend une valeur qui peut etre quelconque
// NB : avant écriture, les emplacements mémoires valent 0xFF

digitalWrite(chipSelectPin, HIGH); // dé-sélectionne l'eeprom

Serial.print("Valeur lue en Eeprom : "),Serial.println(data, DEC);

//while(1);
delay(1000); // entre 2 passages

} // fin de la fonction loop()
```

## Fonctionnement du programme

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



The screenshot shows the Arduino Serial Monitor window titled "/dev/ttyACM0". The window has a text input field at the top with a "Send" button. The main area displays the following output from the program:

```
Etat bit WREN : 1
Millis debut ecriture: 10
0
Millis fin ecriture: 21
0
Valeur lue en Eeprom : 0
```

At the bottom of the window, there is a checked "Autoscroll" checkbox, a dropdown menu set to "Newline", and another dropdown menu set to "115200 baud".



## 20. *Arduino et SPI : en pratique, un dispositif SPI dispose d'une librairie qui gère la communication SPI !*



De ce que nous venons de voir, vous pourriez être découragé !! « La communication SPI... c'est pas pour moi ! » me direz-vous...  
Comprendre le fonctionnement d'un dispositif SPI à partir de sa documentation, envoyer les bons octets de commande, gérer les réponses, etc... Pfiouuu !  
En un sens, je vous comprends : ça n'est pas si simple que ça au final...

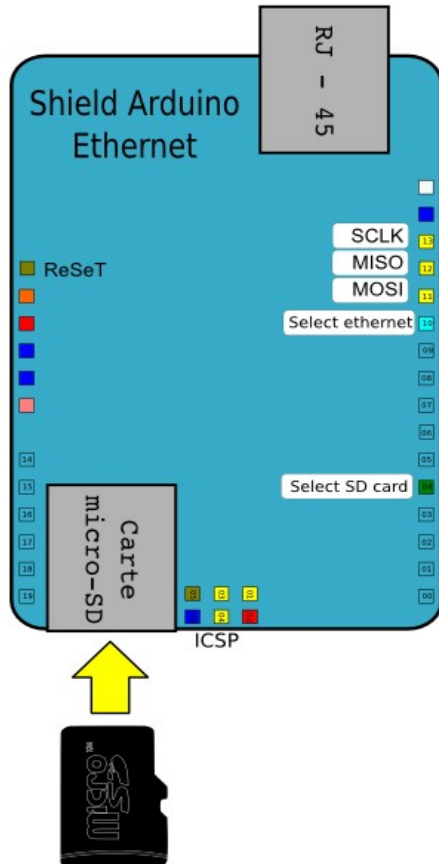
Mais heureusement pour vous, la plupart des dispositifs utilisant la communication SPI utilisent une librairie qui gère la communication directe.  
**Au final, vous n'avez qu'à intégrer à votre programme la librairie SPI ET la librairie du dispositif en question... et c'est tout !**

Ensuite, **vous utilisez directement les fonctions fournies par la librairie du dispositif SPI sans avoir à vous soucier directement de la communication SPI.**

C'est ce que nous allons faire à présent au travers de quelques exemples d'utilisation de dispositifs Arduino utilisant la communication SPI : suivez le guide !



## 21. Le shield Ethernet : description et principe d'utilisation



### Description

- Le module Ethernet Arduino permet à une carte Arduino de se connecter à un réseau Ethernet filaire ou même à internet.
- Ce module intègre également un emplacement pour une carte mémoire micro-SD, pouvant stocker plusieurs Go de données !
- Ce module est basé sur le circuit intégré [Wiznet W5100](#). Le Wiznet W5100 fournit une pile réseau (IP) capable à la fois de **TCP et UDP**. Il supporte jusqu'à quatre connexions simultanées.
- Il suffit d'utiliser la **bibliothèque Ethernet** pour écrire des programmes qui se connectent à un réseau ou à internet en utilisant ce module.

### Brochage

- Ce shield communique avec la carte Arduino en utilisant une communication SPI (voir atelier dédié pour plus de détails). Cette communication utilise les 3 broches suivantes 11 (MOSI), 12 (MISO) et 13 (CLK). *Noter que la connexion de ces broches se fait par le connecteur ICSP de la carte Arduino.*
- La sélection de l'étage utilisé (carte SD ou Ethernet) se fait à l'aide de 2 broches de sélection :
  - la broche 10 pour sélectionner l'étage Ethernet
  - la broche 4 pour sélectionner la carte mémoire SD

**La gestion des broches sera assurée automatiquement par les bibliothèques.**

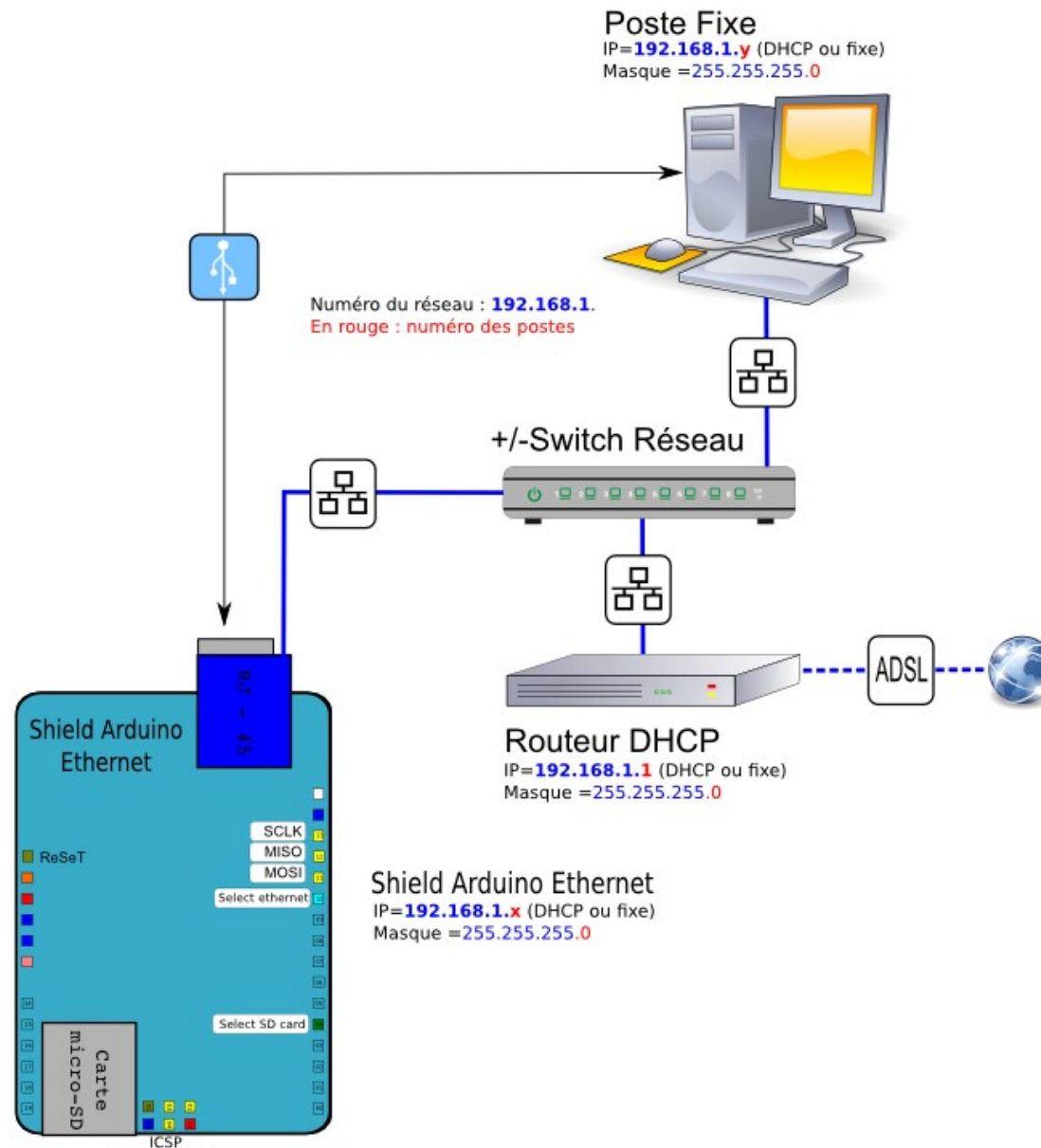
- Toutes les autres broches de la carte Arduino restent disponibles pour d'autres utilisations.

### Principe d'utilisation

- Le module ethernet se connecte « broche à broche » sur une carte Arduino grâce à ses longues broches qui dépassent du circuit imprimé. On dit que le shield est « stackable » !
- Ainsi le brochage de la carte Arduino n'est pas modifié et permet d'enfiler un autre module par dessus et laisse l'accès aux broches de la carte Arduino.

## 22. Faire un simple « ping » vers le shield Ethernet à partir d'un poste fixe : le réseau

- Enfin, nous allons passer à l'action et coder notre premier programme « réseau » pour Arduino . Prêt ? C'est parti.... Voici donc le réseau que nous allons utiliser :



## 23. Programme d'exemple SPI : Initialiser le shield Ethernet

### Ce qu'on va faire ici...

- Je vous propose à titre d'exemple de reprendre un programme présenté dans le tuto d'introduction au réseau Ethernet. Ce programme va être le « programme minimum » pour utiliser le shield Ethernet en place sur la carte Arduino au sein d'un réseau local. On va ici se contenter de configurer l'adresse IP fixe du shield ethernet et ensuite faire un « ping » vers le shield à partir du poste fixe.
- L'intérêt pour nous ici est de montrer comment utiliser la librairie SPI couplée à la librairie Ethernet utilisée avec ce Shield.

Je rappelle une nouvelle fois qu'il est nécessaire d'utiliser la version **Arduino 1.01** (ou suivante) avec les codes qui suivent.

### Entête déclarative

#### Inclusion des librairies utiles

- On commence par inclure les librairies
  - **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
  - et la librairie **Ethernet** qui comporte toutes les fonctions nécessaires pour la communication du shield Ethernet sur le réseau Ethernet local.

#### Déclaration variables et objets utilisés

- On déclare ensuite :
  - un tableau de **byte** correspondant à l'adresse MAC du shield ethernet .

L'adresse MAC est un groupe de nombre hexadécimaux permettant de donner un identifiant unique à chaque matériel utilisé sur un réseau.

- un ou plusieurs objets **IPAddress** correspondant aux différentes adresses IP de configuration utilisée. Ici, nous ne définirons que l'adresse IP locale du shield Ethernet.

```
// --- Inclusion des librairies ---
#include <SPI.h> // librairie SPI - obligatoire avec librairie Ethernet
#include <Ethernet.h> // librairie Ethernet

// --- Déclaration des variables globales ---

//--- l'adresse mac = identifiant unique du shield
// à fixer arbitrairement ou en utilisant l'adresse imprimée sur l'étiquette du shield
byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x1A, 0x71 };

//----- l'adresse IP fixe à utiliser pour le shield Ethernet ---
IPAddress ipLocal(192,168,1,100); // l'adresse IP locale du shield Ethernet
// ATTENTION : il faut utiliser une adresse hors de la plage d'adresses du routeur DHCP
// pour connaître la plage d'adresse du routeur : s'y connecter depuis un navigateur à l'adresse xxx.xxx.xxx.1
// par exemple : sur livebox : plage adresses DHCP entre .10 et .50 => on peut utiliser .100 pour le shield ethernet
```

## Fonction **setup()**

### Initialisation série

- On initialise la connexion série

### Initialisation du shield Ethernet

- On initialise le module Ethernet avec la fonction `Ethernet.begin()`. Bien comprendre que cette fonction initialise simplement le shield Ethernet d'un point de vue matériel. A ce stade, il n'est configuré ni en serveur, ni en client.

Remarquer les différentes possibilités d'initialiser le module avec la fonction `begin()`, notamment la forme DHCP, mais qui utilise 6K de flash supplémentaire. Moi, je vous conseille la forme IP fixe simple.

### Affichage de l'adresse IP du shield Ethernet

- On affiche l'adresse IP attribuée au module. Remarquer que l'instruction `print` supporte l'objet `IPAddress`.

```
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter 1 seule fois au démarrage du programme ---

// ----- Initialisation fonctionnalités utilisées -----

Serial.begin(115200); // Initialise connexion Série

//---- initialise la connexion Ethernet avec l'adresse MAC du module Ethernet, l'adresse IP Locale
//---- +/- l'adresse IP du serveurDNS , l'adresse IP de la passerelle internet et le masque du réseau local

//Ethernet.begin(mac); // forme pour attribution automatique DHCP - utilise plus de mémoire Flash (env + 6Ko)
Ethernet.begin(mac, ipLocal); // forme conseillée pour fixer IP fixe locale
//Ethernet.begin(mac, ipLocal, serverDNS, passerelle, masque); // forme complète

delay(1000); // donne le temps à la carte Ethernet de s'initialiser

Serial.print("L'adresse IP du shield Ethernet est : " );

Serial.println(Ethernet.localIP());

} // fin de la fonction setup()
```

## Fonction **loop()**

- Est laissée vide.

```
void loop(){ // debut de la fonction loop()

} // fin de la fonction loop()
```

## Fonctionnement du programme

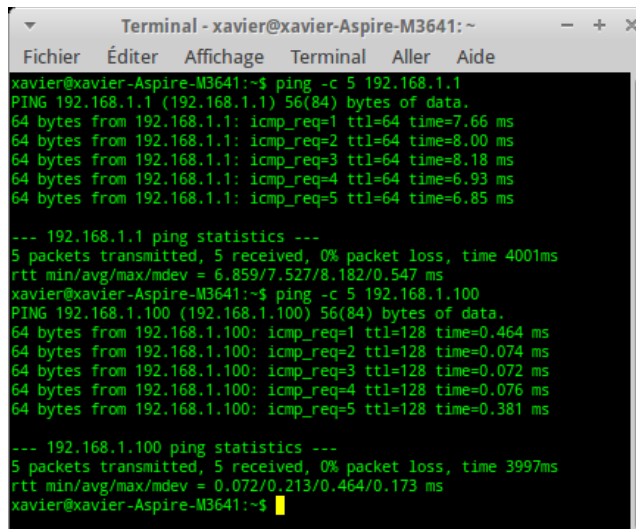
- Une fois la carte Arduino programmée : ouvrir une console système
- Sous Ubuntu (Gnu/Linux) : saisir la commande suivante (cette commande réalise un ping avec 5 envois vers l'adresse indiquée, ici le routeur)

```
ping -c 5 192.168.1.1
```

- ensuite saisir la commande (cette commande réalise un ping avec 5 envois vers l'adresse indiquée, ici le shield Ethernet)

```
ping -c 5 192.168.1.100
```

- Vous devez obtenir quelque chose comme ça dans votre console (ici, remarquer la réponse obtenue du routeur puis du shield Ethernet)



```
Terminal - xavier@xavier-Aspire-M3641:~
Fichier  Éditer  Affichage  Terminal  Aller  Aide
xavier@xavier-Aspire-M3641:~$ ping -c 5 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=7.66 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=8.00 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=8.18 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=6.93 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=6.85 ms

--- 192.168.1.1 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4001ms
rtt min/avg/max/mdev = 6.859/7.527/8.182/0.547 ms
xavier@xavier-Aspire-M3641:~$ ping -c 5 192.168.1.100
PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_req=1 ttl=128 time=0.464 ms
64 bytes from 192.168.1.100: icmp_req=2 ttl=128 time=0.074 ms
64 bytes from 192.168.1.100: icmp_req=3 ttl=128 time=0.072 ms
64 bytes from 192.168.1.100: icmp_req=4 ttl=128 time=0.076 ms
64 bytes from 192.168.1.100: icmp_req=5 ttl=128 time=0.381 ms

--- 192.168.1.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 3997ms
rtt min/avg/max/mdev = 0.072/0.213/0.464/0.173 ms
xavier@xavier-Aspire-M3641:~$
```

**Vous avez réussi ? Bravo, votre réseau local et votre shield Ethernet fonctionnent !**

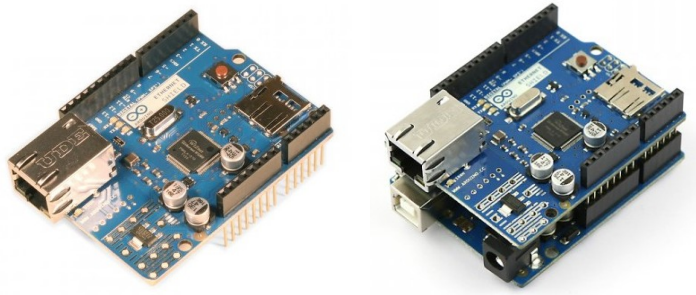
**Et vous avez réussi votre première utilisation de la librairie SPI avec un Shield (=carte d'extension) Arduino !**



## 24. Exemple de shields disposant d'un étage carte SD

Pour l'exemple à suivre, vous aurez idéalement besoin idéalement d'un shield disposant d'un étage carte SD. Plusieurs possibilités, notamment (non exhaustif) :

### Soit d'une carte d'extension (shield) Ethernet (Arduino)



La carte d'extension (ou shield) ethernet Arduino est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser Arduino sur un réseau ethernet local voire même sur internet.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 +/- 4 ) pour communiquer avec Arduino.

**Ce shield intègre également un emplacement pour carte mémoire SD** pour des stockage de données ou de pages HTML locales.

*Ne pas confondre ce shield avec la carte UNO Ethernet qui est une variante d'une carte UNO avec ethernet intégré.*

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

Prix constaté : 33€ environ

### Soit d'une carte d'extension (shield) mémoire SD seule (Seeeduino)



La carte d'extension (ou shield) mémoire SD est une carte électronique enfichable broche à broche sur la carte Arduino et qui permet d'utiliser une carte mémoire micro SD, SD et SDHC.

Ce shield utilise la **communication SPI** (broches 13,12,11, et 10 ) pour communiquer avec Arduino.

disponible chez <http://www.gotronic.fr/> Prix constaté : 12€ environ

### Soit d'une 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

**Noter que de nombreux autres shields disposent d'un étage carte mémoire SD, notamment les shields pour écrans TFT, etc...**

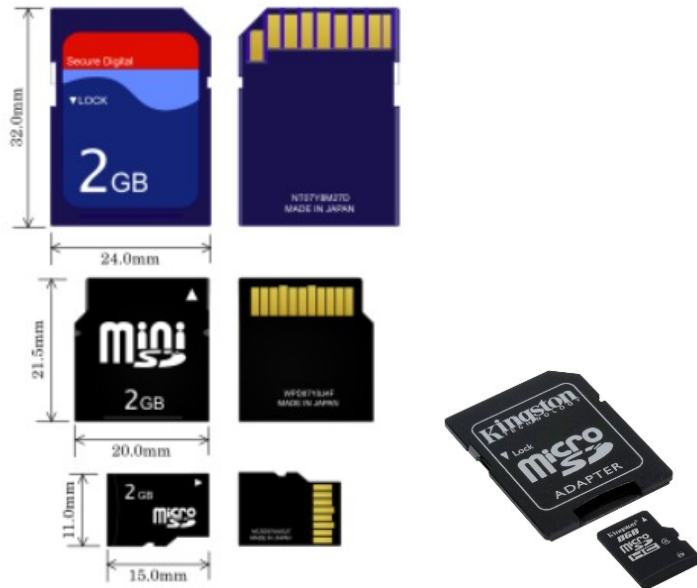
## 25. Pour info : Technique : les cartes mémoires SD

### C'est quoi une carte SD ?

- Je pense que vous le savez : il s'agit d'une carte mémoire permettant de stocker des données de façon non volatile.
- Ces cartes sont très répandues et sont utilisées notamment pour le stockage de photos avec les appareils photos numériques.

### Les différentes dimensions des cartes mémoire SD

- Les cartes mémoires SD existent dans plusieurs tailles différentes, du plus grand au plus petit :
  - le format SD
  - le format mini-SD
  - le format micro-SD
- Noter qu'il existe des adaptateurs micro-SD vers SD (pratiques !)



### Capacités des cartes mémoires SD

- Les SD-Card existent en toutes tailles : 2Go, 4Go, 8Go, 16Go, 32Go, 64Go...
- La véritable différence est le prix. Le bon compromis est un prix inférieur à 1€ / Go
- Avec Arduino, les cartes de plus de 4Go ne sont pas supportées.
- Les catégories de capacité des SD-Card sont indiquées par leur sigle :
  - SD : < 2Go
  - SDHC : 2 à 32 Go
  - SDXC : 32 à 2 To

### Vitesse d'échange d'une carte mémoire SD

- Un point plus méconnu est la vitesse d'échange de la carte SD. Ceci est indiqué par la "classe" de la SD Card, information indiquée sur l'emballage :
  - Les cartes SD classiques sont de classe 4 voire 6.
  - Pour avoir une vitesse d'échange accélérée, préférer une carte classe 10
- Voici les vitesses indicatives pour ces différentes classes, débit minimum garanti (et donc pouvant être supérieur) en écriture, plus élevé en lecture (x1,5 à x2) :
  - classe 4 : 4 Mo/sec
  - classe 6 : 6 Mo / sec
  - classe 10 : 10 Mo/sec

A titre de comparaison, les disques durs SSD actuels ont des vitesses en écriture de l'ordre de 250 Mo/sec et les disques durs classiques jusqu'à 100 Mo/sec.

L'utilisation d'une carte mémoire SD avec Arduino va permettre le stockage de grandes quantités de données (jusqu'à 4 Go à comparer aux 32Ko d'une carte UNO) et va permettre un stockage de données « non-volatiles », le tout sur un support directement utilisable sur un ordinateur classique au besoin.

## 26. Programme d'exemple SPI : communiquer avec une carte mémoire SD

### Ce qu'on va faire ici...

- Nous allons tout simplement ici lire les informations d'une carte SD. Nous reprenons un code présenté par ailleurs dans le tuto dédié à la mémorisation de données avec une carte mémoire SD. Je ne rentre pas ici dans les détails qui sont présentés dans le tuto en question.

### Entête déclarative

#### Inclusion des librairies utiles

- On commence par inclure les librairies
  - **SPI** qui permet au shield Ethernet de communiquer avec la carte Arduino
  - et la librairie **SD** qui comporte toutes les fonctions nécessaires pour la gestion de la carte mémoire SD.

#### Déclaration de variables et objets utilisés

- On déclare ensuite :
  - un objet File
  - un objet représentant la carte mémoire SD
  - et un objet représentant la partition

```
#include <SD.h>
#include <SPI.h>

File myFile;
Sd2Card card;
SdVolume volume;
```

## Fonction **setup()**

### *Initialisation série*

- On initialise la connexion série

### *Initialisation de la carte SD*

- On commence par initialiser la carte SD en précisant la vitesse de communication à utiliser ainsi que la broche de sélection.

#### **Important :**

La broche de sélection est la broche /SS (broche 10) par défaut, sauf dans le cas du shield Ethernet où c'est la broche 4 qui est utilisée.

```
void setup()
{
  Serial.begin(115200); // utiliser le meme debit coté Terminal Serie
  Serial.println("Initialisation de la SD card...");

  pinMode(10, OUTPUT); // laisser la broche SS en sortie - obligatoire avec librairie SD

  if (!card.init(SPI_HALF_SPEED, 10)) {
    Serial.println("Echec initialisation!"); // message port Série
    return; // sort de setup()
  }
  Serial.println("Initialisation reussie !"); // message port Série
}
```

## Fonction **setup()** (suite)

### *Affichage informations sur la carte SD*

- Ensuite à l'aide des fonctions de la librairie native de gestion de la carte SD, on accède aux informations de la carte mémoire SD :
  - tout d'abord le type de la carte
  - puis le type de la partition.

```
// affiche le type de la carte
Serial.print("Type de carte SD : ");
switch(card.type()) {
  case SD_CARD_TYPE_SD1:
    Serial.println("SD1");
    break;
  case SD_CARD_TYPE_SD2:
    Serial.println("SD2");
    break;
  case SD_CARD_TYPE_SDHC:
    Serial.println("SDHC");
    break;
  default:
    Serial.println("Unknown");
}

// Essai ouverture partition qui doit etre FAT16 or FAT32
if (!volume.init(card)) {
  Serial.println("Pas de partition FAT16/FAT32. Reformater la carte SD !");
  return;
}

// affiche la taille de la première partition
uint32_t volumesize;
Serial.print("Partition de type FAT");
Serial.println(volume.fatType(), DEC);
Serial.println();

} // fin setup
```

## Fonction **loop()**

- Est laissée vide.

```
void loop(){ // debut de la fonction loop()

} // fin de la fonction loop()
```

## Fonctionnement du programme

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



**Ce second exemple vous a montré à nouveau comment utiliser une librairie dédiée utilisant la communication SPI**

Il est possible de multiplier les exemples de la sorte, ce qui dépasse cependant le cadre de ce tuto d'introduction à la communication SPI.

Vous aurez l'occasion de voir la communication SPI à l'oeuvre dans de nombreux autres tutos : dans ceux dédiés, notamment, aux afficheurs couleurs TFT, au réseau Ethernet, à la mémorisation sur carte SD, etc...

## 27. *Truc technique : utiliser plusieurs shields SPI utilisant une même broche de sélection...*

- En pratique, on peut être amené à coupler des shields Arduino entre eux... et il se peut que plusieurs étages SPI utilisent la même broche de sélection... ce qui pose problème.
- Par exemple, si on souhaite utiliser le shield Ethernet avec le shield « Mémoire » de chez Snootlab, la broche de sélection 10 est commune à l'étage Ethernet et à l'étage carte mémoire SD du shield mémoire.
- La solution : courber vers l'extérieur la patte de la broche 10 de l'un des 2 shields, par exemple celle du shield mémoire. A l'aide d'un jumper « mâle/femelle » connecter l'embout femelle sur la broche courbée et connecter l'embout mâle dans la broche voulue pour sélectionner la carte SD, par exemple la broche 9 ou toute autre broche libre.
- De cette façon, le shield Ethernet utilise la broche de sélection 10 et le shield mémoire utilise la broche de sélection 9 (ou autre)
- Ensuite, au niveau du code, il suffit d'utiliser la broche 9 (ou autre) pour sélectionner la carte SD du module mémoire et le tour est joué.



### 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 SPI. Les fonctionnalités des shields dédiés (Ethernet, mémoire SD, écran TFT, ...) seront présentés en détail dans des tutos spécifiques.**

## 28. Les éléments du langage Arduino étudiés dans cet atelier

### Librairie SPI

Les instructions de la librairie SPI disponibles sont très logiquement :

- **begin()** : Initialise le bus SPI
- **end()** : Désactive le bus SPI
- **setBitOrder()** : Configure l'ordre des bits transmis en sortie ou en entrée sur le bus SPI
- **setClockDivider()** : Configure le diviseur d'horloge du bus SPI par rapport à l'horloge système.
- **setDataMode()** : Configure le mode de transmission des données sur le bus SPI
- **transfer()** : Transfère un octet sur le bus SPI, aussi bien en envoi qu'en réception.

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](http://www.mon-club-elec.fr/pmwiki_reference_arduino/pmwiki.php?n=Main.ReferenceMaxi)



## **29. A présent, vous devriez être capable :**

Vous devez à présent être capable :

- de comprendre le principe de la communication SPI (Interface Serie pour Périphérique) disponible avec Arduino,
- d'utiliser la librairie SPI du langage Arduino
- d'interfacer votre carte Arduino avec une mémoire externe EEPROM série type AT25HP512 qui utilise le protocole de communication série SPI (Interface Série pour Périphérique).
- d'utiliser la communication SPI avec des shields utilisant la communication SPI

... afin d'être en mesure d'utiliser le protocole de communication SPI avec Arduino.

# Table des matières

Utiliser la communication série SPI (librairie SPI) avec Arduino.

Intro |  
Matériel nécessaire pour les ateliers Arduino |  
Matériel spécifique nécessaire pour cet atelier |  
Exemples de shields Arduino utilisant un étage à communication SPI |  
Technique : Communication SPI : principe général |  
Technique : Communication SPI : principe d'utilisation de plusieurs modules |  
Technique : Principe de mise en oeuvre de la communication SPI |  
Connexions SPI de la carte Arduino |  
Truc technique : utiliser un dispositif SPI avec une carte Arduino Mega |  
Rappel : Langage Arduino : Les bibliothèques |  
Langage Arduino : la bibliothèque SPI pour l'utilisation de la communication série SPI |  
Pour info : Arduino : les rouages internes de la communication SPI |  
Pour info : Ce qui se passe sur le bus SPI pendant un échange « Maître - Esclave » |  
Exemple de dispositifs fonctionnant en SPI |  
Informations techniques importantes à retenir en pratique |  
Rappel : Info technique : les circuits intégrés en boîtier DIL |  
Technique : Composant : Un exemple de composant SPI : une mémoire Eeprom type AT25256 |  
Exemple d'utilisation SPI : Communiquer directement avec une mémoire Eeprom SPI : le montage |  
Exemple d'utilisation SPI : Communiquer directement avec une mémoire Eeprom SPI : le programme |  
Arduino et SPI : en pratique, un dispositif SPI dispose d'une bibliothèque qui gère la communication SPI ! |  
Le shield Ethernet : description et principe d'utilisation |  
Faire un simple « ping » vers le shield Ethernet à partir d'un poste fixe : le réseau |  
Programme d'exemple SPI : Initialiser le shield Ethernet |  
Exemple de shields disposant d'un étage carte SD |  
Pour info : Technique : les cartes mémoires SD |  
Programme d'exemple SPI : communiquer avec une carte mémoire SD |  
Truc technique : utiliser plusieurs shields SPI utilisant une même broche de sélection... |  
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](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ATELIERS)