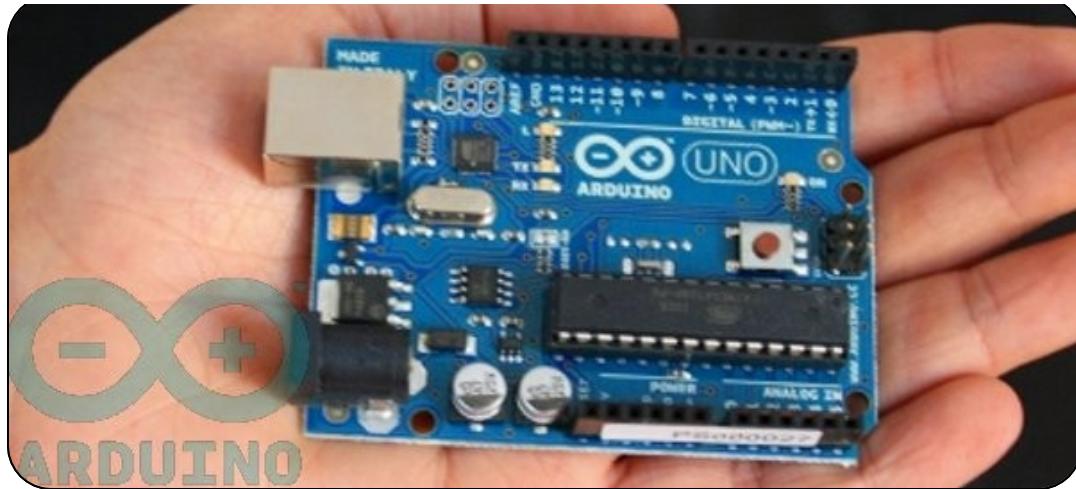


Université  
de ToulouseUniversité  
de Toulouse

# MICRO-CONTROLEURS & OPEN SOURCE HARDWARE



Jérémie GRISOLIA

Département de Génie Physique – INSA TOULOUSE

05.61.55.96.58 – Bureau 138 (1er étage)

[jeremie.grisolia@insa-toulouse.fr](mailto:jeremie.grisolia@insa-toulouse.fr)

<http://moodle.insa-toulouse.fr/course/view.php?id=494>



Laboratoire  
de Physique & Chimie  
des Nano-Objets



Après un bref rappel de ce qu'est un microcontrôleur, de son architecture interne et de ses possibilités, je présenterai l'environnement Arduino pour finalement réaliser une plateforme Open-Source Hardware.

L'Arduino® est une plateforme complète de développement électronique permettant de réaliser à moindre coût des applications performantes à base de micro-contrôleurs.



Ce cours fournit tous les éléments nécessaires à la conception et la mise en œuvre de nombreuses applications avec notamment:

- Des éléments pour créer ses propres circuits (du schématique au routage pour tirer ses PCB)
- La présentation de la syntaxe de langage de programmation et de son environnement de développement;
- Plusieurs dizaines de schémas d'interfaces avec les dispositifs les plus divers (afficheurs, moteurs, capteurs, bus I<sup>2</sup>C, Nunchuk, écran tactile de DS...).
- Plusieurs dizaines d'exemples de programmes types permettant la mise en œuvre de ces interfaces
- Des éléments pour créer vos propres bibliothèques
- Des éléments pour créer vos propres interfaces
- Des éléments d'interfaçage Arduino/Processing (programmation Java) et ANDROID
- Des éléments pour créer une application IoT

## I – LES MICROCONTROLEURS ET LEURS ARCHITECTURES

## II – LA PLATEFORME OPEN-SOURCE ARDUINO®:

- II-1 Qu'est-ce qu'un Arduino ?
- II-2 La plateforme de développement IDE
- II-3 Quels sont les composants adressables: actionneurs et capteurs ?

## III – MISE EN ŒUVRE DES ARDUINO:

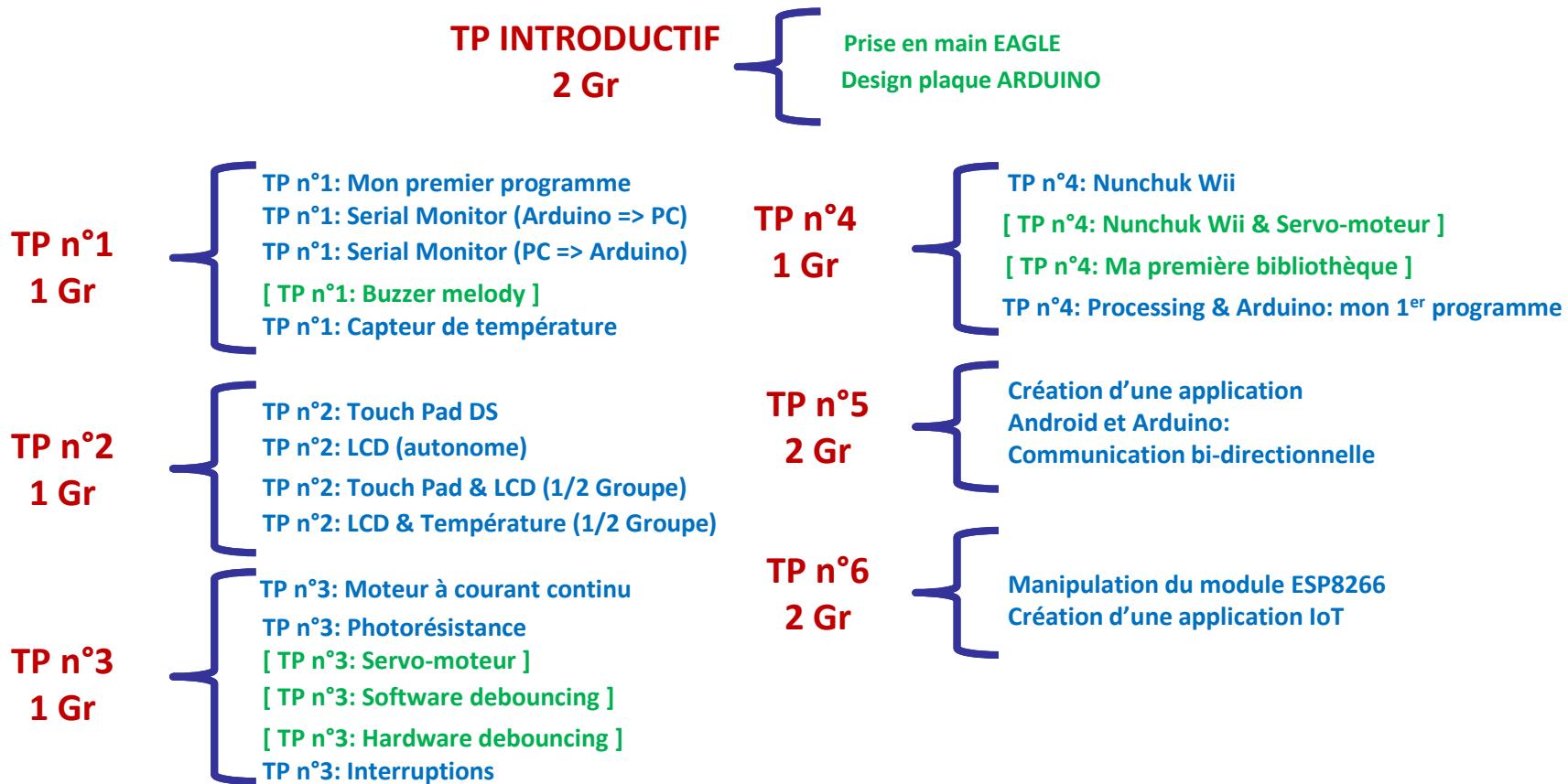
- III-1 les entrées/sorties digitales
- III-2 les entrées/sorties analogiques
- III-3 applications digital & analogique
- III-4 faire de l'analogique avec du digital
- III-5 déparasitage ou debouncing
- III-6 les interruptions (matérielles et logicielles)
- III-7 liaisons séries: asynchrone (RS232) & synchrone (I<sup>2</sup>C, SPI, one wire)
- III-8 créer une librairie
- III-9 les shields & leur création

## IV – COMMUNICATION DE L'ARDUINO AVEC D'AUTRES PLATEFORMES:

- IV- 1 PROCESSING => JAVA,
- IV- 2 ANDROID, PYTHON, FLASH, MXP, PUREDATA...
- IV-3 IoT

## V - LES REFERENCES INDISPENSABLES

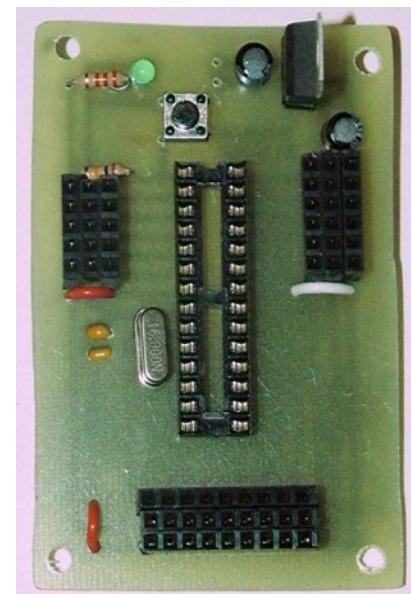
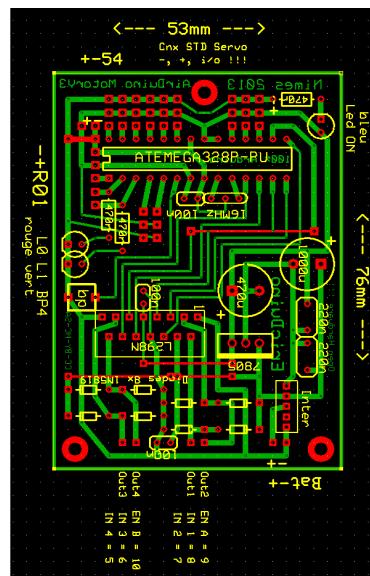
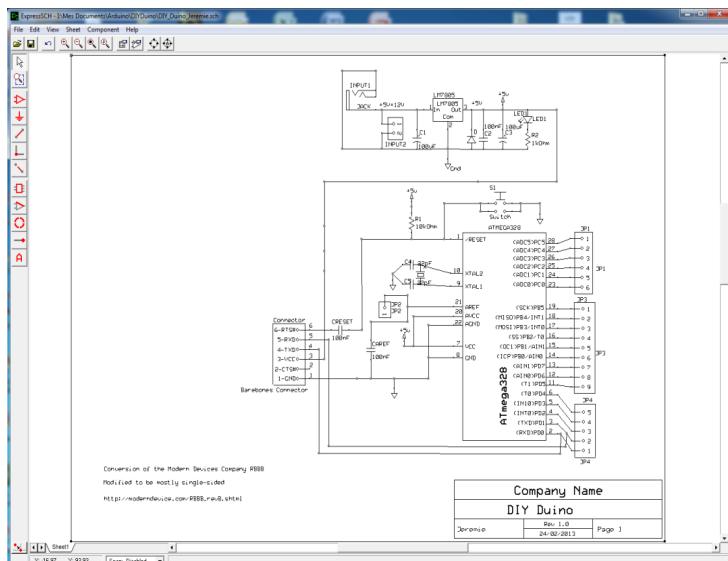
# PLAN DETAILLE DES TRAVAUX PRATIQUES



# EVALUATION DU MODULE DE FORMATION « M-OSH »

**Vous développerez un projet à base d'Arduino (ATMEGA + USB)**

**1 – Vous concevez un shield pour Arduino : schématique + layout + PCB avec EAGLE (**/5 : /2.5 schématique + routage, /2.5 PCB + montage**) : *Attention au deadline à déposer sur le MOODLE !***



**2 - Vous concevez le(s) programme(s) (Arduino et e.g. Androïd) qui va(vont) avec la plateforme PCB réalisée (**/5**)**

**3 – Vous publierez les éléments de votre projet sur le Wiki du cours:**

<http://moodle.insa-toulouse.fr/course/view.php?id=494> (**/5**)

**4 – Vous soutiendrez votre projet devant un jury d'enseignants et d'industriels (e.g. Snootlab, AIRBUS...) (**/5**)**

You will make an Arduino shield to interface a robotic arm with a small gripper and teach function. The robotic arm will be manufactured using FABRIC'INSA Fablab facilities. This robot can follow, can learn and can repeat endless the teached movements.

**Teach mode:** After a reset the robot arm follows the teach in arm while simple mapping the analog inputs every 25ms to the servo motors.

Pressing the button stores each servo position in a array

**Play mode:** Double press the button switch to play mode.

The sketch reads the array step by step and moves the robot arm.

For cool locking movements, a routine calculates different micro steps for each servo to have moving start and end sync on all axis.

Also added a ramp for soft increase/decrease velocity.

Shorter travel distances the robot does slow, longer distances with faster speed.

Its all about timing so my thoughts in this moment

Ressources: <http://letsmakerobots.com/robot/project/micro-servo-robot>

Micro Servo Robot : <https://www.youtube.com/watch?v=bLnAJ-mSEIE> \*\*\* (avec Arduino et mémoire)

<http://letsmakerobots.com/robot/project/micro-servo-robot>

[https://www.youtube.com/watch?v=Gx5be7\\_J2xc](https://www.youtube.com/watch?v=Gx5be7_J2xc)

Bras robotique "Braccio" : <http://www.lextronic.fr/P37761-bras-robotique-braccio.html>

Building a 6-axis Robot Arm : [https://www.youtube.com/watch?v=cod\\_SNq-d6c](https://www.youtube.com/watch?v=cod_SNq-d6c)

## 1 - Le matériel « hardware » est « open source » :

Le Matériel Libre (OSHW - OpenSource Hardware) est un terme qui regroupe des produits « tangibles » — machines, appareils ou tous dispositifs physiques — dont les plans ont été rendus publics de telle manière que quiconque puisse les **fabriquer, modifier, distribuer et les utiliser (sous les mêmes conditions que la licence de l'œuvre originale)**.



## 2 - Le logiciel « software » est « open source » :

On peut l'utiliser, le modifier et le distribuer librement  
(sous les mêmes conditions que la licence de l'œuvre originale).



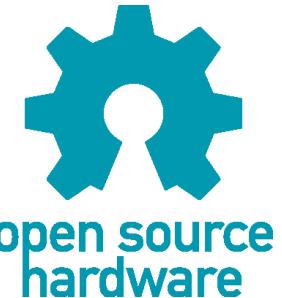
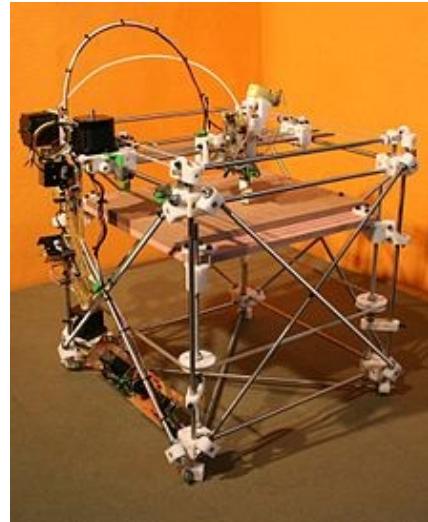
Attention, en OSH, la conception du matériel (i.e. dessins mécaniques, schémas, nomenclatures, layout PCB, le code source et layout de circuits intégrés...), EN PLUS du logiciel qui pilote les matériels sont tous tenus à la démarche des logiciels libres (free and open-source software (FOSS))

*La plupart des auteurs, quel que soit leur champ d'activité, et indépendamment de leur statut d'amateur ou de professionnel, ont tout intérêt à favoriser un écosystème dans lequel les œuvres peuvent être diffusées, réutilisées et dérivées de manière créative.*

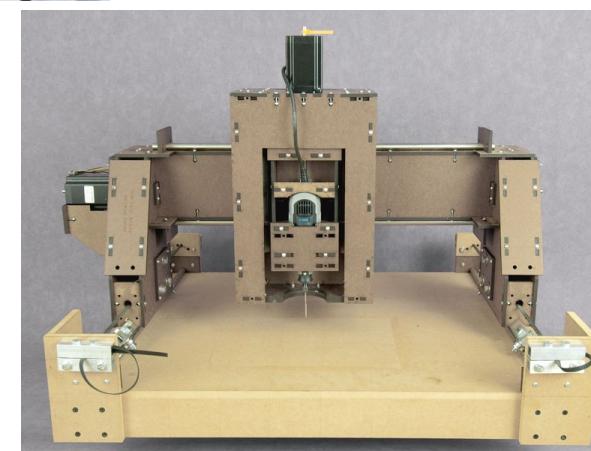
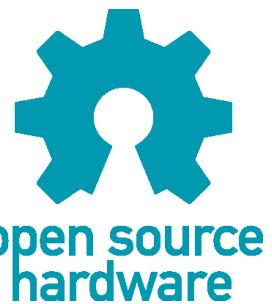
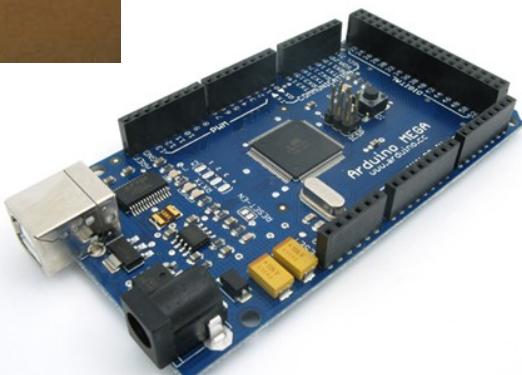
*Plus il est facile de réutiliser et de dériver des œuvres, plus notre culture s'enrichit.*

<http://freedomdefined.org/OSHW/translations/fr>  
[http://en.wikipedia.org/wiki/Open\\_source\\_hardware](http://en.wikipedia.org/wiki/Open_source_hardware)  
<http://freedomdefined.org/Definition/Fr>

# EXEMPLES D'OPEN-SOURCE HARDWARE



REPRAP



-2-

February 3, 1976

An Open Letter to Hobbyists

To me, the most critical thing in the hobby market right now is the lack of good software courses, books and software itself. Without good software and an owner who understands programming, a hobby computer is wasted. Will quality software be written for the hobby market?

Almost a year ago, Paul Allen and myself, expecting the hobby market to expand, hired Monte Davidoff and developed Altair BASIC. Though the initial work took only two months, the three of us have spent most of the last year documenting, improving and adding features to BASIC. Now we have 4K, 8K, EXTENDED, ROM and DISK BASIC. The value of the computer time we have used exceeds \$40,000.

The feedback we have gotten from the hundreds of people who say they are using BASIC has all been positive. Two surprising things are apparent, however. 1) Most of these "users" never bought BASIC (less than 10% of all Altair owners have bought BASIC), and 2) The amount of royalties we have received from sales to hobbyists makes the time spent of Altair BASIC worth less than \$2 an hour.

Why is this? As the majority of hobbyists must be aware, most of you steal your software. Hardware must be paid for, but software is something to share. Who cares if the people who worked on it get paid?

Is this fair? One thing you don't do by stealing software is get back at MITS for some problem you may have had. MITS doesn't make money selling software. The royalty paid to us, the manual, the tape and the overhead make it a break-even operation. One thing you do do is prevent good software from being written. Who can afford to do professional work for nothing? What hobbyist can put 3-man years into programming, finding all bugs, documenting his product and distribute for free? The fact is, no one besides us has invested a lot of money in hobby software. We have written 6800 BASIC, and are writing 8080 APL and 6800 APL, but there is very little incentive to make this software available to hobbyists. Most directly, the thing you do is theft.

What about the guys who re-sell Altair BASIC, aren't they making money on hobby software? Yes, but those who have been reported to us may lose in the end. They are the ones who give hobbyists a bad name, and should be kicked out of any club meeting they show up at.

I would appreciate letters from any one who wants to pay up, or has a suggestion or comment. Just write me at 1180 Alvarado SE, #14, Albuquerque, New Mexico, 87108. Nothing would please me more than being able to hire ten programmers and deluge the hobby market with good software.

*Bill Gates*Bill Gates  
General Partner, Micro-Soft

[http://fr.wikipedia.org/wiki/An\\_Open\\_Letter\\_to\\_Hobbyists](http://fr.wikipedia.org/wiki/An_Open_Letter_to_Hobbyists)

**Définition de Hobbyist**

En anglais, *hobby* veut dire « passe-temps, [loisir](#) » ; un *hobbyist* est une personne qui « bricole ». Dans sa lettre, les *hobbyists* à qui s'adresse Gates bricolent des [ordinateurs personnels](#) – la grande nouveauté à l'époque car jusqu'alors, l'informatique était confinée au monde scientifique et étudiant. Ce nouveau marché n'est pourtant pas [grand public](#) car il faut toujours de solides compétences en électronique et en programmation pour pouvoir utiliser ces ordinateurs. On pourrait remplacer « *hobbyist* » par « amateur d'informatique personnelle ».

Ces *hobbyists* utilisent des *hobby software*, des « logiciels personnels » que l'on peut utiliser chez soi ; ce marché est lui aussi un marché émergeant car jusqu'alors les logiciels étaient développés par/pour les scientifiques et nécessitaient la puissance d'un [mainframe](#).

Mainframe utilisés par Bill Gates pour découvrir l'informatique à la [Lakeside School](#) de [Seattle](#). Il y réalise avec [Paul Allen](#) son premier [programme](#), un jeu de [tic-tac-toe](#).

[Richard Stallman](#), initiateur du mouvement pour le [logiciel libre](#) (terme utilisé par opposition à celui de [logiciel propriétaire](#)), fait aussi ses premiers pas en informatique sur ce type de machine.

Les *hobbyists* n'ayant pas les compétences requises ou tout simplement le temps de développer leurs applications sont les clients de ce marché.

L'idée d'un « [marché du logiciel](#) » s'oppose aux pratiques mises en place par les universitaires durant les années 1950-60 : les constructeurs ne fournissent que le matériel, c'était aux utilisateurs de développer leurs propres outils. Ces outils étaient généralement partagés gracieusement entre universitaires. Les exemples le plus flagrant sont le [système d'exploitation ITS](#) et le logiciel [Emacs](#).

**La lettre ouverte**

David Bunnell, rédacteur en chef de *Computer Notes*, comprend la position de Gates et écrit, en septembre 1975, qu'« une partie des clients de MITS ont volé leurs logiciels »<sup>19</sup>.

« Maintenant, posez-vous ces questions : est-ce qu'un musicien est en droit d'encaisser une partie des recettes de la vente de son enregistrement ? Est-ce qu'un écrivain peut recevoir une partie de la recette des ventes de son livre ? Est-ce que les personnes qui copient un logiciel sont différentes de celles qui copient un album ou un livre<sup>21</sup>?<sup>20</sup> »

— David Bunnell, *Computer Notes* n°4

Dans sa lettre ouverte, Gates poursuit la même idée que Bunell en septembre, et que Roberts en octobre : « Comme la majorité des hobbyists devraient le savoir, la plupart d'entre-vous volent leurs logiciels. Le matériel doit être acheté, mais les logiciels sont un chose que l'on partage. Qui se soucie de savoir si les personnes qui ont travaillé dessus seront payées ? »<sup>22</sup>.

Une copie de la lettre de Gates est envoyé au [Homebrew Computer Club](#), la cible principale. La lettre apparaît aussi dans *Computer Notes* et Dave Bunnell l'envoie par service exprès aux principales parutions en informatique du pays<sup>23</sup>.

**Réactions**

La lettre connaît de nombreuses réactions : certains pensent que les logiciels devraient être inclus avec la machine et donc que la méthode de [distribution](#) utilisée par Gates n'est pas adaptée ; d'autres posent des questions sur le coût réel de développement.



- Le **copyleft** est l'autorisation donnée par l'auteur d'un travail soumis au droit d'auteur (œuvre d'art, texte, programme informatique ou autre) d'utiliser, d'étudier, de modifier et de copier son œuvre, dans la mesure où cette autorisation reste préservée.

=> l'auteur refuse que son travail puisse évoluer avec une restriction du droit à la copie (copyright).

- De ce fait, le contributeur apportant une modification (correction, ajout, réutilisation, etc.) est contraint de redistribuer ses propres contributions avec les mêmes conditions d'utilisation que l'original.

*Autrement dit, les créations réalisées à partir d'éléments sous copyleft héritent de facto de ce copyleft.*

La licence libre la plus connue utilisant le copyleft est la licence publique générale GNU mais il existe aussi d'autres licences, spécifiquement créées pour certains domaines très divers (art, jeu de rôle, revue scientifique, etc.), qui peuvent être considérées comme des « licences copyleft ».



**La licence libre Copyleft est la meilleure façon d'encourager le partage et de favoriser la créativité.**  
La création intellectuelle doit se libérer des entraves du copyright, parce que le copyright ne sert que l'industrie, non la culture. **La culture ne peut fleurir que dans le partage !**

*Si quelqu'un s'approprie mon travail, l'améliore et en fait quelque chose de nouveau, de beau, de grand, d'utile, via le Copyleft, alors j'en serais très heureux et tout le monde sera gagnant.*

*Bref, je crois que la copie n'est pas un frein, mais un levier.*

<http://fr.wikipedia.org/wiki/Copyleft>

<http://artlibre.org/licence/lal>

[http://des-trucs-pour-changer-de-vie.blogspot.fr/2013/02/des-trucs-pour-changer-de-vie-devient.html#.UUw2Pjc\\_4tI](http://des-trucs-pour-changer-de-vie.blogspot.fr/2013/02/des-trucs-pour-changer-de-vie-devient.html#.UUw2Pjc_4tI)

Les licences Creative Commons ont été créées en partant du principe que la propriété intellectuelle était fondamentalement différente de la propriété physique, et du constat selon lequel les lois actuelles sur le copyright étaient un frein à la diffusion de la culture.



#### BUT:

Fournir un outil juridique qui garantit à la fois la protection des droits de l'auteur d'une œuvre artistique et la libre circulation du contenu culturel de cette œuvre, ceci afin de permettre aux auteurs de contribuer à un patrimoine d'œuvres accessibles dans le « domaine public » (notion prise au sens large).



Cette licence comprenant bien que :

**Renonciation** — N'importe laquelle des conditions ci-dessus peut être levée si vous avez l'autorisation du titulaire de droits.

**Domaine Public** — Là où l'œuvre ou un quelconque de ses éléments est dans le domaine public selon le droit applicable, ce statut n'est en aucune façon affecté par la licence.

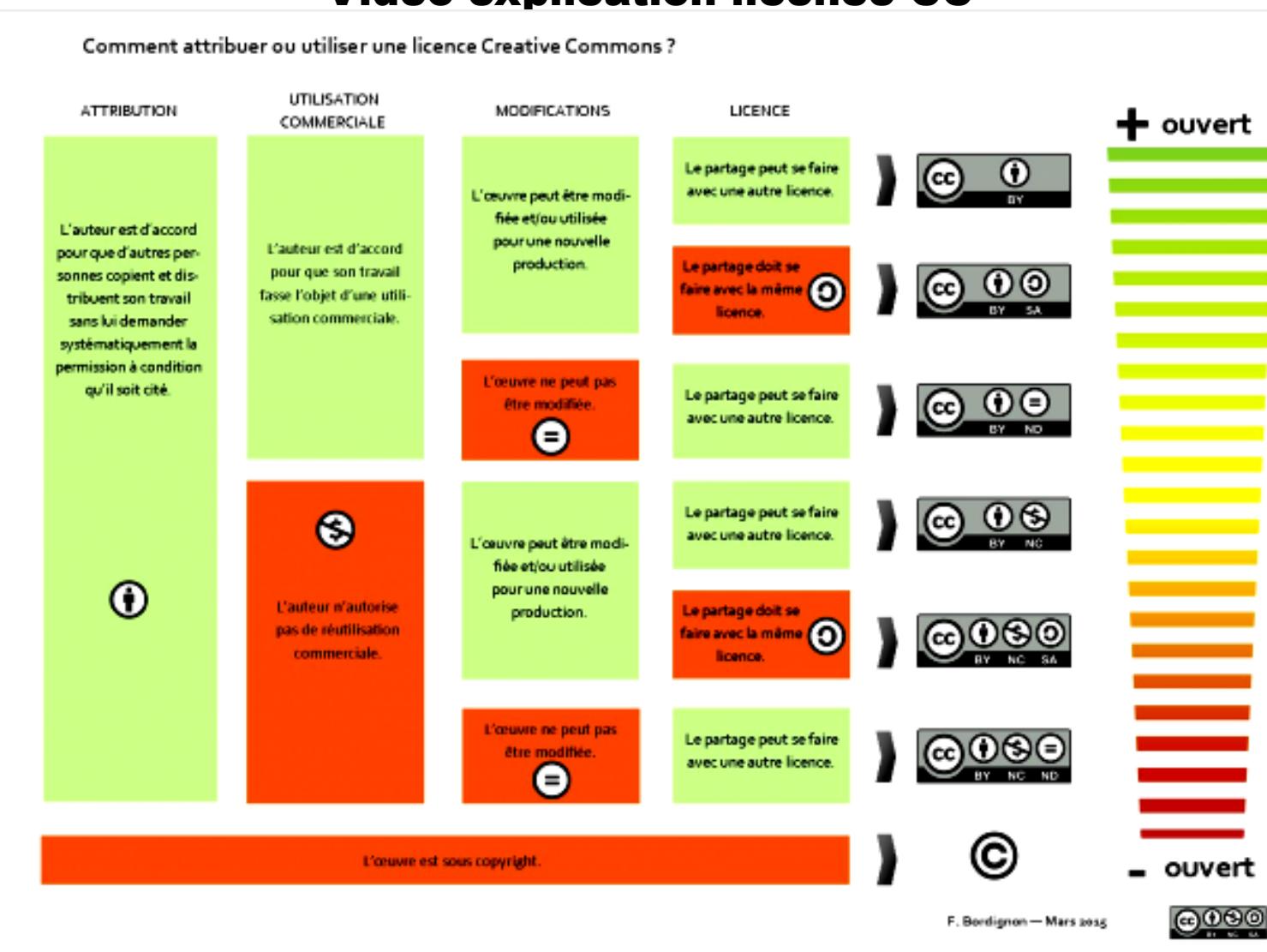
<http://creativecommons.org/licenses/by/3.0/deed.fr>

[http://fr.wikipedia.org/wiki/Licence\\_Creative\\_Commons](http://fr.wikipedia.org/wiki/Licence_Creative_Commons)

# CLASSIFICATION DES LICENCES

## Video explication licence CC

Comment attribuer ou utiliser une licence Creative Commons ?



F. Bordignon — Mars 2015



<http://www.netpublic.fr/2015/04/comment-choisir-une-licence-creative-commons/>  
<http://www.netpublic.fr/2012/12/creative-commons-video/>  
<http://www.netpublic.fr/2012/03/creative-commons-dossier/>

# CLASSIFICATION DES LICENCES

Désignation complète du contrat	Terme abrégé désignant la licence	Symboles désignant la licence				Type de licence
Paternité	CC-BY					<a href="#">Licence libre non copyleft</a>
Paternité Partage des conditions initiales à l'identique	CC-BY-SA					<a href="#">Licence libre copyleft</a>
Paternité Pas de modification	CC-BY-ND					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale	CC-BY-NC					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale Partage des conditions initiales à l'identique	CC-BY-NC-SA					<a href="#">Licence de libre diffusion</a>
Paternité Pas d'utilisation commerciale Pas de modification	CC-BY-NC-ND					<a href="#">Licence de libre diffusion</a>



- **Paternité [BY] (Attribution)** : l'œuvre peut être librement utilisée, à la condition de l'attribuer à l'auteur en citant son nom.



- **Pas d'utilisation commerciale [NC] (Noncommercial)** : le titulaire de droits peut autoriser tous les types d'utilisation ou au contraire restreindre aux utilisations non commerciales (les utilisations commerciales restant soumises à son autorisation).



- **Pas de modification [ND] (NoDerivs)** : le titulaire de droits peut continuer à réservé la faculté de réaliser des œuvres de type dérivées ou au contraire autoriser à l'avance les modifications, traductions.



- **Partage des conditions initiales à l'identique [SA] (ShareAlike)** : le titulaire des droits peut autoriser à l'avance les modifications ; peut se superposer l'obligation (SA) pour les œuvres dites dérivées d'être proposées au public avec les mêmes libertés (sous les mêmes options Creative Commons) que l'œuvre originale.

# I - LES MICROCONTROLEURS

# DEFINITIONS

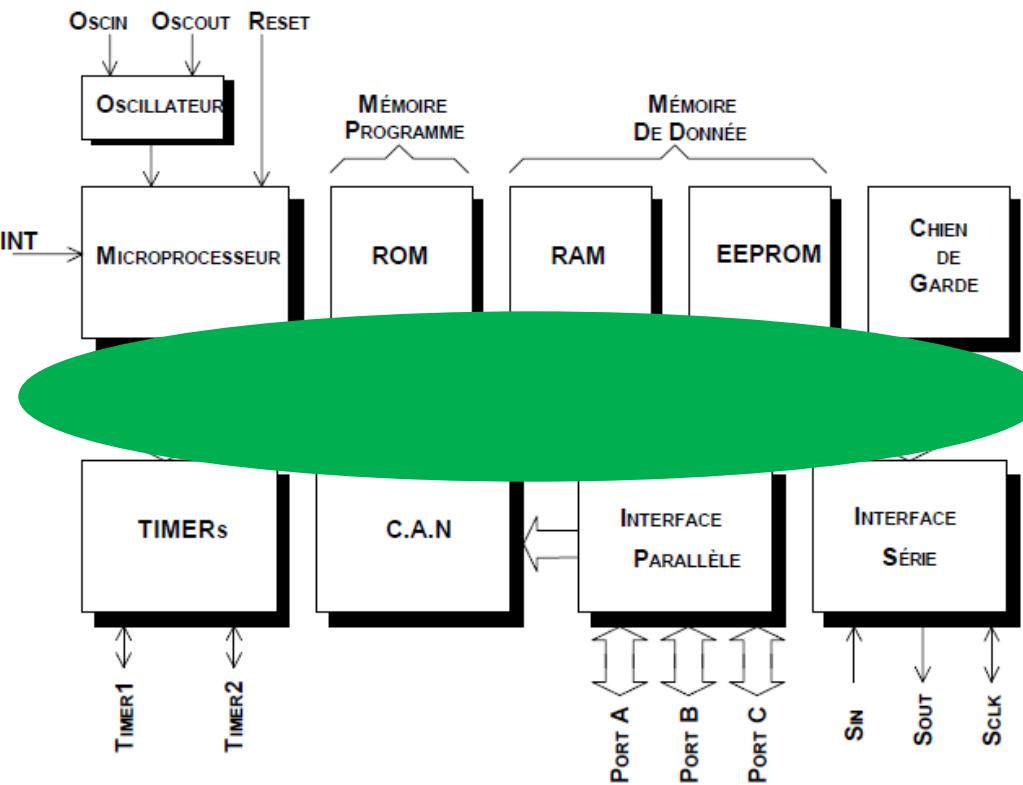
## DÉFINITION:

Un microcontrôleur est un circuit intégré rassemblant dans un même boîtier **TOUTES** les fonctions électroniques d'un système: un microprocesseur, plusieurs types de mémoires (mémoire vive, mémoire non-volatile) et des périphériques de communication (Entrées-Sorties Analogiques/Digitales).

=> la microélectronique entre alors dans l'ère du **SLI (System Level Integration)**

*Attention: il est appelé à tort MICROPROCESSEUR par les personnes mal informées voire même par certains journalistes « scientifiques » ;-)*

- Un microprocesseur (C.P.U.),
- De la mémoire de donnée (RAM et EEPROM),
- De la mémoire programme (ROM, ...),
- Des interfaces parallèles pour la connexion des entrées/sorties,
- Des interfaces séries (synchrone ou asynchrone) pour le dialogue avec d'autres unités,
- Des timers pour générer ou mesurer des signaux avec une grande précision temporelle,
- Des convertisseurs analogique/numérique pour le traitement de signaux analogiques.
- Une structure recevant l'oscillateur pour cadencer le µproc
- Et des bus de communications: adresse, données et contrôles qui relie ces différents sous-ensembles organes



## 1- C.P.U. (MICROPROCESSEUR): le microprocesseur a besoin de certains éléments pour fonctionner :

- de la mémoire morte dite ROM: principalement pour stocker le programme,
- de la mémoire vive dite RAM: principalement pour stocker les variables,
- des périphériques (principalement pour interagir avec le monde extérieur),
- une horloge pour le cadencer (principalement à quartz).

Il exécute séquentiellement les instructions stockées dans la mémoire programme.

Il est capable d'opérer sur des mots binaires dont la taille, en bits, est celle du bus des données (*parfois le double pour certains microcontrôleurs*).

Il est généralement constitué des éléments suivants :

- Différents registres contenant les opérandes, les résultats des opérations, le mode d'adressage...
- Un compteur programme pointant l'adresse de la prochaine instruction à exécuter
- Une unité arithmétique et logique (ALU) permettant d'effectuer des opérations entre l'accumulateur et une opérande: chargée des calculs +, -, \*, /, AND, OR, NOT

On peut noter qu'il existe 2 catégories opposées de microprocesseurs: les CISC et les RISC.

- **CISC (Complex Instruction Set Computer)**: ce microprocesseur possède un nombre important d'instructions. Chacune d'elles s'exécute en plusieurs périodes d'horloges.
- **RISC (Reduced Instruction Set Computer)**: ce microprocesseur possède un nombre réduit d'instructions. Chacune d'elles s'exécute en une période d'horloge.

En 1975, David Patterson (University of California, Berkeley):

80% des instructions d'un programme n'utilisent que 20% des instructions du CPU

Idée: enlever les instructions qui ne servent pas et grâce au gain de place rajouter des éléments utiles (par exemple de la mémoire , des registres...)

=> CHOIX STRATEGIQUE pour raccourcir le temps CPU

$$CPU\_time = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} * \frac{\text{avg\_cycles}}{\text{instructions}} * \frac{\text{seconds}}{\text{avg\_cycles}}$$

1 - Les CISC raccourcissent le temps d'exécution en réduisant le nombre d'instructions par programme

⇒ un processeur coûteux (CISC) car complexe et qui chauffe beaucoup

2 - Les RISC raccourcissent le temps d'exécution en réduisant le nombre de cycles d'horloge par instruction (i.e. des instructions simples prennent moins de temps à interpréter)

=> un processeur économique (RISC) car simple et qui consomme peu... mais dont le programme nécessite plus d'instructions et donc plus de mémoire.

- De plus, avec moins de transistors le RISC chauffe moins => peut monter plus facilement en fréquence.

Les transistors économisés servent à créer des mémoires ou des registres (le PowerPC RISC en a 2 ou 3x plus qu'un PC (CISC)), ce qui évite d'accéder trop souvent à la mémoire (ralentissant le processeur).

Au final: le CISC diminuent la vitesse de traitement, mais les instructions sont plus complexes, plus puissantes, et plus nombreuses

- Au contraire, le RISC consiste à déplacer les complexités majeures du hardware vers le software,

⇒ Un CISC permet de faire un petit programme complexe,

⇒ Un RISC permet de faire un gros programme complexe.

# RISC VS CISC

Fragment de programme (multiplier deux entiers) :

**CISC**

```
mov ax, 10
mov bx, 5
mul bx, ax
```

$$10 * 5 = 50$$

**RISC**

```
Begin    mov ax, 0
         mov bx, 10
         mov cx, 5
         add ax, bx
loop     Begin
```

$$10 + 10 + 10 + 10 + 10 = 50$$



**RISC 1 / CISC 0**

Le nombre de cycle total pour le CISC :  $(2 \text{ movs} \times 1 \text{ cycle}) + (1 \text{ mul} \times 30 \text{ cycles}) = 32 \text{ cycles}$

Le nombre de cycle total pour le RISC :  $(3 \text{ movs} \times 1 \text{ cycle}) + (5 \text{ adds} \times 1 \text{ cycle}) + (5 \text{ loops} \times 1 \text{ cycle}) = 13 \text{ cycles}$

Que se passe-t-il pour un multiplication de  $10 * 500000$  ?

**RISC 1 / CISC 1**

$\Rightarrow$  pas si simple !

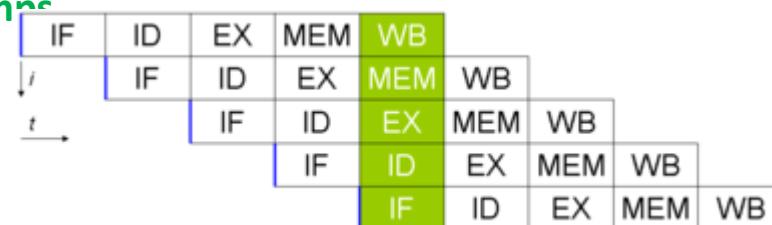
Malheureusement, la technologie RISC a aussi des défauts:

En effet, le nombre réduit d'instructions fait que le software résultant, à fonctions à accomplir égales, occupe plus de mémoire que celui d'une machine CISC, aussi bien statiquement que dynamiquement.

Toutes les machines RISC utilisent la technique du "pipelining" pour augmenter leurs prestations en terme d'instructions exécutées dans l'unité de temps



Séquençage des instructions dans un processeur **sans pipeline**. Il faut 15 cycles pour exécuter 3 instructions.



Séquençage des instructions dans un processeur **dôté d'un pipeline** à 5 étages. Il faut 9 cycles pour exécuter 5 instructions. À  $t = 5$ , tous les étages du pipeline sont sollicités, et les 5 opérations ont lieu en même temps.

EN RESUME:

-Il n'y a pas de vainqueur !

⇒ cela dépend du contexte :

CISC	RISC
Used in laptops and desktop computers, made by Intel or AMD	Used in smartphones and tablets, based around ARM processor
Has more complex hardware	Has simpler hardware
Multiple machine cycles per instruction	Single machine cycle per instruction
Physically larger in size and require more silicon to make thus more expensive	Smaller in size as less complex circuitry required, less silicon needed to make thus cheaper
Greater energy consumption	Lower energy requirements, and can go into "sleep mode" when not actively processing
More intensive tasks will do better with CISC	Run at lower clock speed, but can perform simpler tasks more quickly than CISC
Can't support pipelining	Can support pipelining

- Dans l'architecture CISC, utilisée notamment dans la gamme des x86 d'AMD, puis INTEL, Cyrix..., les concepteurs misent sur la réduction du nombre d'instructions nécessaires pour exécuter le programme, en concevant des instructions très puissantes, ce qui a l'inconvénient de devoir augmenter en moyenne le nombre de cycles machine nécessaires pour compléter une instruction. Dans ce cas, la fréquence de travail du système est réduite car il faut introduire une phase d'interprétation du code machine à travers des microcodes.

- Par contre, dans l'architecture RISC, on mise beaucoup sur la minimisation du nombre des cycles machine et l'on rend la majeure partie des instructions exécutables en un seul cycle d'horloge, ce qui permet d'augmenter la fréquence de travail du système.

- Ceci est possible en éliminant la phase d'interprétation grâce à la simplicité des instructions qui peuvent être décodées et exécutées directement par une simple unité de contrôle câblée. La simplification des unités de contrôle des machines de type RISC est particulièrement avantageuse pour la réalisation de la CPU sur un seul chip VLSI. L'économie d'espace obtenue permet, à zone de silice égale, d'augmenter sensiblement le nombre de registres internes et/ou d'intégrer directement sur le chip la mémoire cache pour exploiter au maximum la vitesse du microprocesseur.

## 2.2 MÉMOIRES PROGRAMMES:

Ce dispositif contient les instructions du programme que doit exécuter le microprocesseur.

Ce type de mémoire, **appelée mémoire morte**, est uniquement accessible en lecture.

Sa programmation nécessite une procédure particulière et un matériel adéquat.

**Il en existe différents types selon leur mode de programmation :**

- **De la ROM** dont le contenu est programmé lors de sa fabrication.
- **De la PROM programmable** électriquement une seule fois par le développeur (appelée aussi OTPROM),
- **De la EPROM programmable** électriquement et effaçable aux U-V (appelée aussi UVeprom),
- **De la EEPROM programmable** et effaçable électriquement.
- **De la FLASH**, beaucoup plus rapide que les autres.

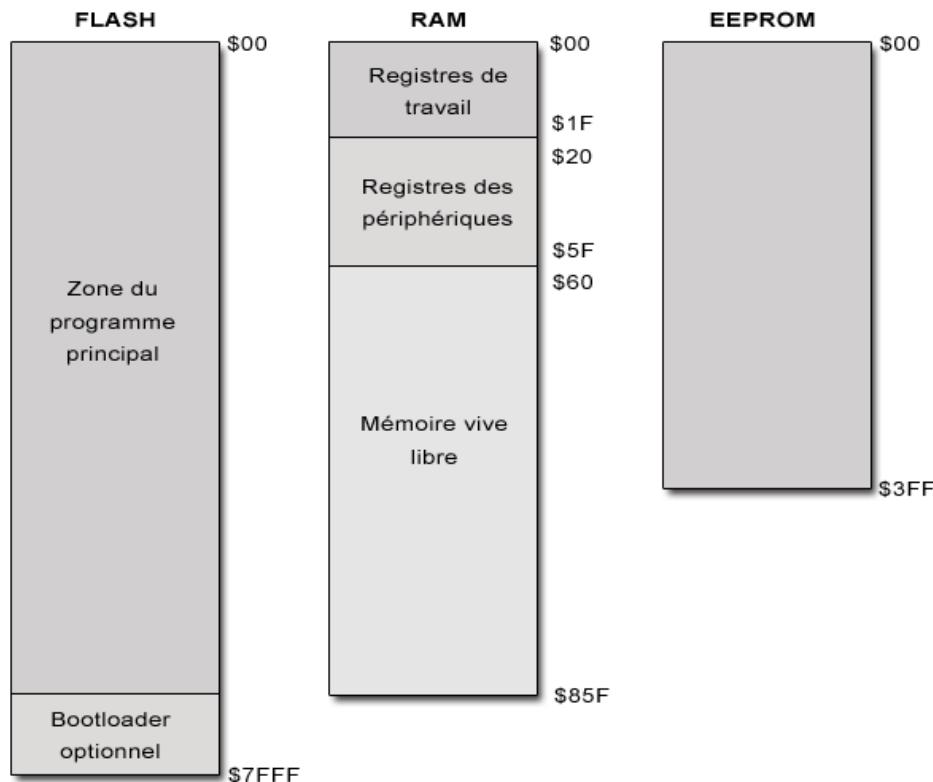
## 2.3 MÉMOIRES DE DONNÉES:

Ce dispositif permet de mémoriser temporairement les données générées par le microprocesseur pendant les différentes phases du traitement numérique (résultats d'opérations, états des capteurs...).

**Ces mémoires sont accessibles en écriture et en lecture.**

**On en trouve 2 types :**

- **De la mémoire vive (RAM) volatile** (données perdues en cas de coupure de l'alimentation) ayant un temps de lecture et écriture assez court (quelques ns),
- **De la mémoire morte (EEPROM) non-volatile** (données conservées en cas de coupure de l'alimentation) ayant un temps d'écriture assez élevé (quelques ms) par rapport au temps de lecture qui est assez faible (quelques ns).



Par exemple: L'ATMEGA32 est doté de 3 types de mémoires principales :

**La FLASH :** mémoire qui garde son contenu lorsqu'elle n'est plus sous tension. Capacité est de 32 Ko (\$00 à \$7FFF).

A la fin de cette espace mémoire => espace prédéfini au **BOOTLOADER**:

petit morceau de programme qui s'exécute avant votre propre programme afin d'effectuer diverses opérations.

**La RAM :** mémoire qui perd son contenu quand elle n'est plus alimentée. Capacité de 2 Ko (\$00 à \$85F).

Permet de :

- mémoriser différentes informations temporaires lors de l'exécution d'un programme,
- contient l'adresse de retour lors d'un appel à un sous programme ou d'une routine d'interruption.

**L'EEPROM :** mémoire qui conserve les données lors de coupure de courant. Capacité de 1 Ko (\$00 à \$3FF).

Elle permet de mémoriser des informations qui doivent être rechargées à la mise sous tension.

Eeprom (Electrically Erasable Programmable Read Only Memory)

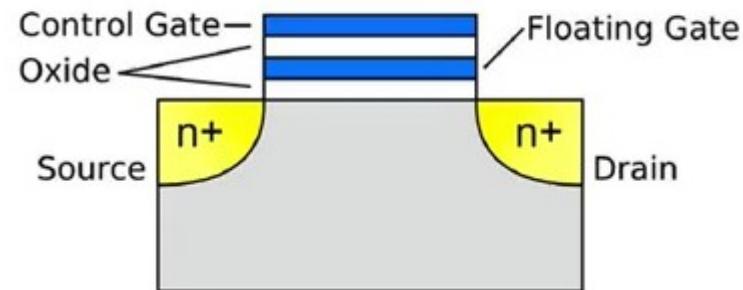
Taille 64 – 2k bytes

Durée de rétention > 10 ans

Cycle d'écriture lent (3-10ms)

nombre de cycles limités (100'000 à 10<sup>6</sup>)

Cycle de lecture normal



## 2.4 L'INTERFACE PARALLÈLE:

Ce type d'interface, répartie sur plusieurs ports (maximum 8 bits), permet de prendre en compte:

- des états logiques appliqués en entrée (**état de capteurs**) ou de
- générer des signaux binaires en sortie (**commande d'actionneurs**).

Les broches de ces ports peuvent donc être **configurées en entrée ou en sortie**, avec différentes options (résistances de rappel, sorties collecteurs ouverts, interruption...).

La configuration ainsi que l'état logique de ces broches est obtenue par des opérations d'écriture ou de lecture dans différents registres associés à chaque port.

On trouve généralement :

- Un registre de direction pour une configuration en entrée ou en sortie,
- Un registre de donnée recopiant les états logiques de chaque broche de port,
- Un registre d'option permettant plusieurs configurations en entrée ou en sortie.

## 2.5 L'INTERFACE SÉRIE:

Ce type d'interface permet au microcontrôleur de communiquer avec d'autres systèmes à base de microprocesseur. Les données envoyées ou reçues se présentent sous la forme d'un succession temporelle (sur un seul bit) de valeurs binaires images d'un mot.

Il y a 2 types de liaison série : ASYNCHRONE ET SYNCHRONE

### Ø Liaison série ASYNCHRONE:

Ce dispositif ne possède pas de signal d'horloge de synchronisation.

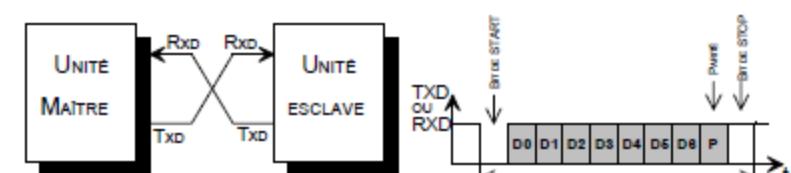
Les unités en liaison possèdent chacune une horloge interne cadencée à la même fréquence.

Lorsqu'une unité veut émettre un mot binaire, elle génère un front descendant sur sa ligne émettrice.

A la fin de l'émission de ce mot, la ligne repasse au niveau haut.

La donnée à transmettre peut contenir un bit supplémentaire appelé "parité" et servant à la correction d'erreurs.

Exemple: votre liaison favorite RS232 gérée par un UART ou USART (Universal Asynchronous Receiver Transmitter) qui est un émetteur-récepteur asynchrone universel, (qui possède un tampon de 128 caractères dans l'Arduino 7bits + 1 bit parité), et qui réalise aussi, par le biais d'un convertisseur USB-série, la liaison vers le PC.



### Ø Liaison série SYNCHRONE:

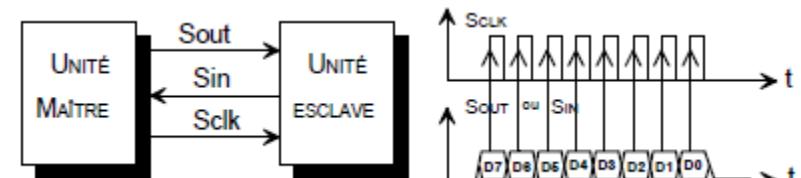
Dans ce dispositif la transmission est synchronisé par un signal d'horloge émis par l'unité maître, conjointement au signal véhiculant les données.

Exemples:

SPI (Serial Peripheral Interface Bus) par Motorola => three wires

I<sup>2</sup>C (Inter Integrated Circuit) par Philips ~1980 => two wires

One Wire (protocole DALLAS de MAXIM) => one wire



## 2.6 LE CAN: convertisseur Analogique/Numérique:

Le CAN intégré dans les microcontrôleurs est généralement du type "Approximations successives".

Il possède plusieurs entrées multiplexées accessibles via les broches des ports de l'interface parallèle.

Le CAN possède normalement 2 registres :

- Un registre de données contenant le résultat de la conversion,
- Un registre de contrôle permettant de lancer et de surveiller la conversion.

## 2.7 LE TIMER: les Timers permettent de réaliser les fonctions suivantes :

- Génération d'un signal périodique modulé ou non en largeur d'impulsion (PWM),
- Génération d'une impulsion calibrée,
- Temporisation,
- Comptage d'événements.

Plusieurs registres associés aux Timers permettent de configurer les différents modes décrits ici.

## 2.8. LE CHIEN DE GARDE (Watch Dog): est un système anti-plantage du microcontrôleur, qui s'assure qu'il n'y ait pas d'exécution prolongée d'une même suite d'instruction.

Exemple: si le programme attend le résultat d'un système extérieur (e.g. du CAN) ou si le programme tourne dans une boucle sans fin => aucune réponse => BLOCAGE.

Dans la pratique: on remet à zéro périodiquement (à intervalle définissable) un registre interne grâce à l'instruction **clrwdt** (clear watchdog). Si le programme est bloqué, il ne passe plus dans la branche de remise à zéro et le comptage va jusqu'au bout, déclenche le watchdog qui relance le programme.

Le watchdog des microcontrôleurs AVR est temporisé par une horloge interne à 1 MHz, il peut donc fonctionner également en l'absence de l'horloge du système car il est indépendant de celui-ci.

## Les registres des périphériques :

Ces registres permettent :

- d'accéder à la configuration des périphériques intégrés (USART, CAN, SPI,...) et timers, EEPROM...
- et d'obtenir des états sur leurs fonctionnement.

## Timers 8 et 16 bits

Par exemple: le tableau ci-contre énumère le nom des registres de l'ATMEGA32

## Mémoire

## Interfaces parallèles des E/S

## Liaisons Série

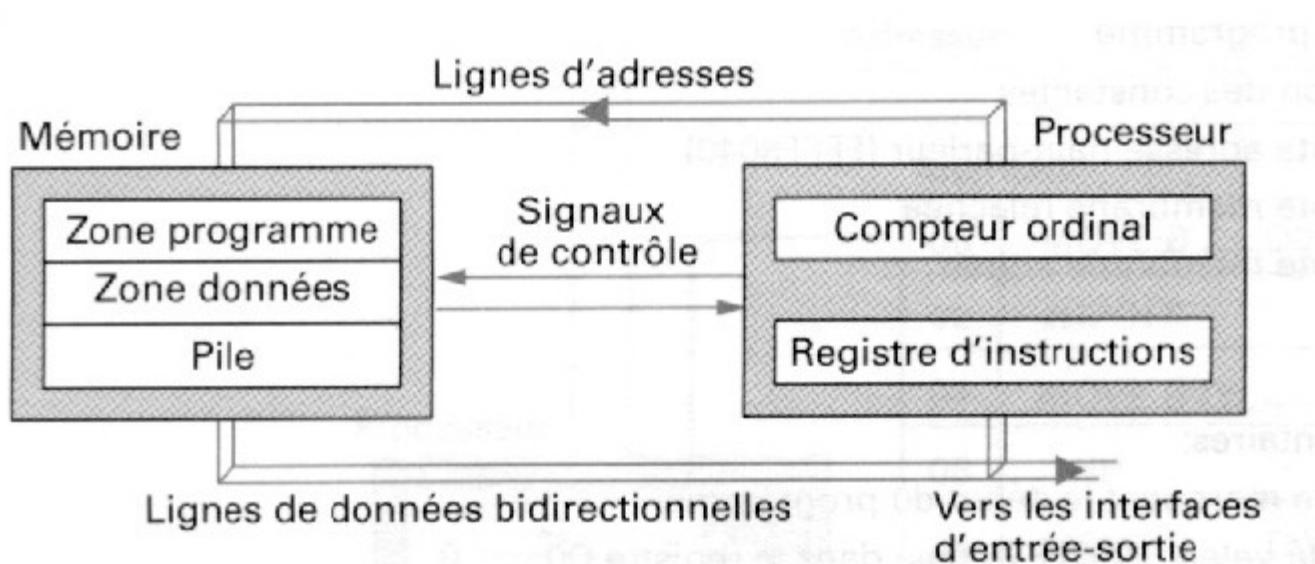
## Convertisseur A/D

<http://www.atmicroprog.com/cours/atmega/regsystem.php>

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
\$3F (\$5F)	SREG	I	T	H	S	V	N	Z	C
\$3E (\$5E)	SPH	—	—	—	—	SP11	SP10	SP9	SP8
\$3D (\$5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
\$3C (\$5C)	OCR0	Timer/Counter0 Output Compare Register							
\$3B (\$5B)	GICR	INT1	INT0	INT2	—	—	—	IVSEL	IVCE
\$3A (\$5A)	GPIO	INT1	INT0	INTF2	—	—	—	—	—
\$39 (\$59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
\$38 (\$58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
\$37 (\$57)	SPMCR	SPMIE	RWWBSB	—	RWWBSRE	BLBSET	PGWRT	PGERS	SPMEN
\$36 (\$56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	—	TWIE
\$35 (\$55)	MUCUR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
\$34 (\$54)	MUCUSR	JTD	ISC2	—	JTRF	WDRF	BORF	EXTRF	PORF
\$33 (\$53)	TCCR0	EOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
\$32 (\$52)	TCNT0	Timer/Counter0 (8 Bits)							
\$31 <sup>(1)</sup> (\$51) <sup>(1)</sup>	OSCAL	General Selection Register							
	ODCR	On-Chip Debug Register							
\$30 (\$50)	SFIOR	ADTS2	ADTS1	ADTS0	—	ACME	PUD	PSR2	PSR10
\$2F (\$4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
\$2E (\$4E)	TCCR1B	ICNC1	ICES1	—	WGM13	WGM12	CS12	CS11	CS10
\$2D (\$4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte							
\$2C (\$4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte							
\$2B (\$4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte							
\$2A (\$4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte							
\$29 (\$49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte							
\$28 (\$48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte							
\$27 (\$47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte							
\$26 (\$46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte							
\$25 (\$45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
\$24 (\$44)	TCNT2	Timer/Counter2 (8 Bits)							
\$23 (\$43)	OCR2	Timer/Counter2 Output Compare Register							
\$22 (\$42)	ASSR	—	—	—	—	AS2	TCN2UB	OCR2UB	TCR2UB
\$21 (\$41)	WDTCR	—	—	—	WDTDE	WDE	WDP2	WDP1	WDP0
\$20 <sup>(2)</sup> (\$40) <sup>(2)</sup>	UBRRH	URSEL	—	—	—	UBRR[11:8]			
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USB5	UCS1	UCS0	UCPOL
\$1F (\$3F)	EEARH	EEPROM Address Register Low Byte							
\$1E (\$3E)	EEARL	EEPROM Address Register High Byte							
\$1D (\$3D)	EEDR	EEPROM Data Register							
\$1C (\$3C)	EECR								
\$1B (\$3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
\$1A (\$3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
\$19 (\$39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0
\$18 (\$38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
\$17 (\$37)	DDRB	DBB7	DBB6	DBB5	DBB4	DBB3	DBB2	DBB1	DBB0
\$16 (\$36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
\$15 (\$35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
\$14 (\$34)	DDRC	DCD7	DCD6	DCD5	DCD4	DCD3	DCD2	DCD1	DCD0
\$13 (\$33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
\$12 (\$32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
\$11 (\$31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
\$10 (\$30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
\$0F (\$2F)	SPDR	SPI Data Register							
\$0E (\$2E)	SPSR	SPIE	WDR	—	—	—	—	—	SPI2X
\$0D (\$2D)	SPCR	CPOL	CPH	DDR0	MSTR	CPOL	CPHA	SPR1	SPR0
\$0C (\$2C)	UDR	USART I/O Data Register							
\$0B (\$2B)	UCSRA	IAC	TAC	UDRE	FE	DOR	PE	U2X	MPCM
\$0A (\$2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCS2	RXB8	TXB8
\$09 (\$29)	UBRR	USART Baud Rate Register Low Byte							
\$08 (\$28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0
\$07 (\$27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
\$06 (\$26)	ADCWSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
\$05 (\$25)	ADCCH	ADC Data Register High Byte							
\$04 (\$24)	ADCCL	ADC Data Register Low Byte							
\$03 (\$23)	TWDR	Two-wire Serial Interface Data Register							
\$02 (\$22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE
\$01 (\$21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	—	TWPS1	TWPS0
\$00 (\$20)	TWBR	Two-wire Serial Interface Bit Rate Register							

Les différents blocs sont reliés par 3 bus :

- le **bus d'adresse** qui permet au microprocesseur de sélectionner la case mémoire ou le périphérique auquel il veut accéder pour lire ou écrire une information (instruction ou donnée) ;
- le **bus de données** qui permet le transfert des informations entre les différents blocs ; ces informations seront soit des instructions, soit des données en provenance ou à destination de la mémoire ou des périphériques ;
- le **bus de contrôle** qui indique si l'opération en cours est une lecture ou une écriture, si un périphérique demande une interruption etc.



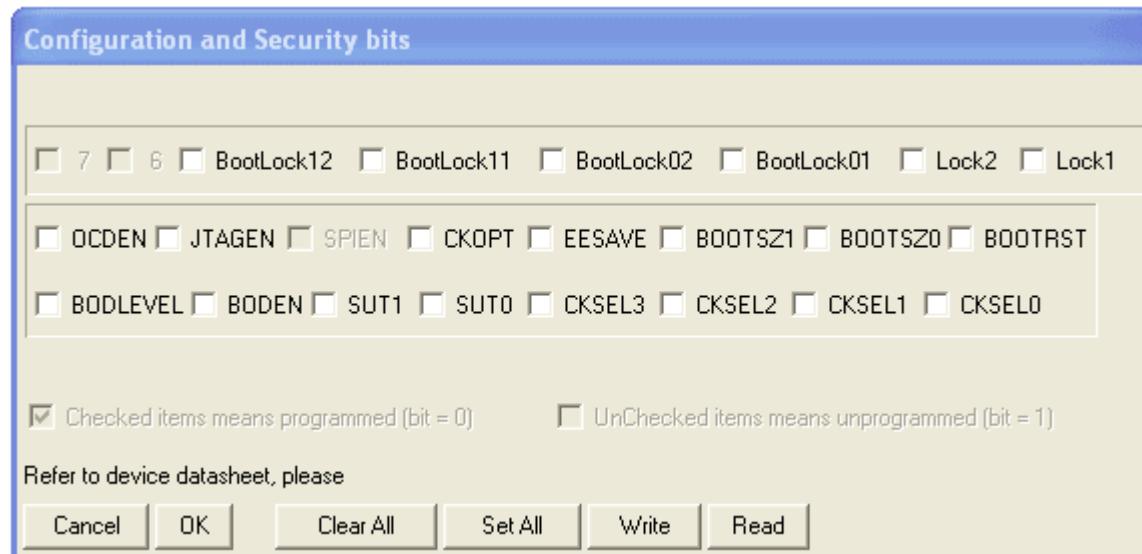
Les fusibles ont toujours existé ne serait ce que pour la protection contre le téléchargement de programme afin d'empêcher la duplication ou le désassemblage:  
bref de quoi verrouiller le programme de toute curiosité industrielle.

Par exemple: la classe ATMEGA possède 21 Fusibles permettant de multiples réglages : source et vitesse d'horloge, détecteur de sous-tension (brown-out)...

Jargon utilisé par Atmel :

**Un fusible programmé a pour valeur "0"**  
**Un fusible non programmé a pour valeur "1"**

On peut donc retenir que la logique intellectuelle est inversée.



Copie d'écran des fusibles sur un ATMEGA32 avec le logiciel Pony Prog

# LES FUSIBLES

## LES FUSIBLES DE PROTECTION:

**Bootlock2, Bootlock1** : paire de fusibles destinés à contrôler les droits d'accès aux mémoires FLASH et EEPROM.

Bootlock2	Bootlock1	Type de protection
1	1	Aucune restriction d'accès aux mémoires.
1	0	Aucune programmation possible de la FLASH ou de l'EEPROM par programmation série et parallèle. La programmation des fusibles est désactivée
0	0	Aucune programmation et vérification possible de la FLASH ou de l'EEPROM par programmation série et parallèle. La programmation des fusibles est désactivée

## MODE OSCILLATEUR RC INTERNE CALIBRÉ:

Utilise l'oscillateur interne de type RC à des vitesses pré-définies de 1, 2, 4, 8 MHz. Ces fréquences sont données pour une tension d'alimentation de 5V et 25°C.

CKOPT	CKSEL3	CKSEL2	CKSEL1	CKSELO	Fréquence (MHz)
1	0	0	0	1	1
1	0	0	1	0	2
1	0	0	1	1	4
1	0	1	0	0	8

## MODE OSCILLATEUR EXTERNE À QUARTZ:

Mode le plus utilisé.

**CKOPT** permet de sélectionner 2 modes différents:

1 - Si programmé (0): rail to rail, soit la capacité maximum.

2 - Si non programmé (1) : le circuit d'oscillation consommera moins d'énergie mais aura une fréquence de fonctionnement limitée.

*ATMEL préconise de programmer ce fusible pour f=16 MHz*

CKOPT	CKSEL3	CKSEL2	CKSEL1	Plage de Fréquence (MHz)	Capacité oscillation recommandée
1	1	0	1	0.4 - 0.9 *	-
1	1	1	0	0.9 - 3.0	12 - 22 pF
1	1	1	1	3.0 - 8.0	12 - 22 pF
0	101,110,111			1.0 <=	12 - 22 pF

**JEU D'INSTRUCTIONS:** on peut classer les instructions qu'un microcontrôleur est capable d'effectuer en quelques groupes.

## 1 - Instructions de transfert

Le microcontrôleur passe une grande partie de son temps à transférer des octets d'un endroit à l'autre du système :

- d'un périphérique vers un registre interne ou vice-versa,
- d'un registre interne vers la mémoire RAM ou vice-versa.

**Attention:** *un transfert direct d'une case mémoire vers une autre ou vers un périphérique, ou une écriture en mémoire ROM, ne peut en général pas être fait, la structure du microcontrôleur rend obligatoire le passage des informations par un de ses registres internes.*

*Remarquons que, sauf exceptions, il s'agit plutôt d'une copie que d'un transfert puisque la case mémoire d'origine garde son information (tant qu'on n'a pas écrit autre chose à la place).*

## 2 - Instructions arithmétiques

Un microcontrôleur, surtout un 8 bits, n'est pas un grand mathématicien.

**Il est seulement capable d'effectuer des additions, des soustractions, des multiplications et des divisions sur des nombres binaires de 8 bits.**

Toutes les opérations mathématiques complexes telles que le traitement des grands nombres, des nombres fractionnaires, des puissances, des racines carrées, des fonctions trigonométriques, logarithmiques et exponentielles doivent être ramenées à une succession d'opérations simples portant seulement sur des octets. *Des routines mathématiques (petits programmes permettant de réaliser les calculs complexes) ont été développées pour la plupart des microcontrôleurs populaires.*

### 3 - Instructions logiques:

Les microcontrôleurs sont capables d'effectuer des opérations logiques : **ET, OU, XOU (XOR), NON (inverseur), rotations, décalages...**

Les opérations sont opérées **simultanément sur les bits correspondant des deux registres.**

**Exemple: comparer deux octets A et B selon une opération logique, c'est réaliser une soustraction dont on néglige le résultat ;**

on veut simplement savoir s'elle est nulle (ce qui signifie que  $A = B$ ), positive ( $A > B$ ) ou négative ( $A < B$ ).

Ces indications sont inscrites dans des indicateurs d'états (petites mémoires d'1 bit situées dans le processeur).

### 4 - Instructions d'entrées/sorties, utilisées pour :

- lire l'état d'un port d'entrée (permettant l'interfaçage d'interrupteurs, de commutateurs, d'optocoupleurs, de convertisseurs analogiques/numériques, de claviers, etc.) ;
- écrire une information dans le registre d'un port de sortie, qui maintient l'information à la disposition des circuits extérieurs (leds, moteurs, relais, convertisseurs numériques/analogiques, etc.)
- écrire ou lire une information dans les registres d'un port série.

## -5 - Instructions de branchement: instructions qui altèrent le déroulement normal du programme.

- On distingue les sauts et les sous-routines:
- Les sauts provoquent un branchement du programme vers une adresse mémoire qui n'est pas contigüe à l'endroit où l'on se trouve.
- La sous-routine ou sous-programme est une partie de programme dont on a besoin à plusieurs endroits dans l'exécution du programme principal. Plutôt que de répéter ce sous-programme à tous les endroits où l'on en a besoin, on le place en un endroit donné (par exemple à la fin du programme principal) et on opère un branchement du programme principal vers le sous-programme chaque fois que nécessaire. La grande différence par rapport au saut, c'est qu'au moment du branchement il faut mémoriser l'adresse d'où l'on vient, afin de pouvoir y revenir une fois le sous-programme terminé. Ceci est effectué en mémorisant l'adresse de départ dans un registre ad hoc (la pile) du microcontrôleur.

Tant les sauts que les sous-routines peuvent être :

- inconditionnels ;
- conditionnels, c'est-à-dire que le branchement n'a lieu que si une certaine condition est remplie

## 6 - Instructions diverses:

- des instructions de gestion de la pile (zone de mémoire RAM permettant le stockage de données pendant l'exécution du programme) ;
- des instructions de contrôle du processeur : par exemple passage en mode basse consommation, contrôle des périphériques embarqués (i.e. sur la même puce que le processeur) ;
- des instructions permettant de positionner des indicateurs internes du processeur.

Ces instructions varient fort selon les familles des microcontrôleurs.

Compte tenu de l'intégration de tous ces éléments dans un seul et unique boîtier de circuit intégré, il ne faut que très peu de composants électronique externes autour du microcontrôleur pour le faire fonctionner:

**1 - L'alimentation:** tous les µcontrôleurs actuels fonctionnent sous une tension de 1.8 à 6V avec un préférence pour le 5V (en raison l'abondance des circuits logiques de la famille dite TTL qui utilisent cette tension) . Bientôt cependant => 3.3V.

Très important: la tension doit être stabilisée pour qu'il fonctionne correctement.

**2 – L'horloge:** un µcontrôleur est un circuit logique séquentiel, i.e. qui fonctionne au rythme d'un signal rectangulaire, appelé horloge qui cadence toute sa circuiterie interne.

Gamme: quelques kHz à plusieurs dizaines de MHz (en comparaison dans un PC on dépasse les GHz !!!)

**Remarque: plus f↑, plus il est rapide mais plus il consomme et plus il chauffe !!!**

*Attention: il est recommandé d'avoir une horloge stable => pilotée par un quartz ou un résonateur céramique, seuls composants électroniques capables de générer des signaux à fréquence stable et précise.*

**3 – Le circuit de reset:** tout comme un PC, il exécute en permanence un programme, et si le programme plante, il faut pouvoir sortir de cette situation => le reset fait recommencer le programme sans débrancher l'alimentation.

3 grandes étapes:

1 - Les premiers microcontrôleurs datent des années 1970:  
(Texas TMS1000 et Intel 4048, avec une mask-ROM



Mask-ROM: mémoire masquée, écrite directement dans le Silicium.

Il fallait:

- 1 - écrire un programme,
- 2 - le transmettre au fabricant et quelques mois plus tard:
- 3 – Recevoir un certain nombre d'exemplaires de notre circuit intégré
- 4 - *Espérer que notre programme soit juste et qu'il marche !!!!*

2 - EPROM: Erasable Programmable Read Only Memory (avec effacement par Ultra-Violets): Intel 8748, Motorola 68705, etc...



3 - EEPROM: dès les PIC 16c84 (en 1993)



**8 bits:**

- PIC Microchip
- AVR Atmel
- Dérivés de 80C51, 80C52

**16 bits:**

- dsPIC
- MSP430 Texas Instruments

**32 bits:**

- AVR32, PIC 32, MIPS, Power PC

• ARM produits par de nombreux fabricants: NxP, STMicro, Texas, Samsung, Infineon, Toshiba, Analog Device, Qualcomm, Freescale...

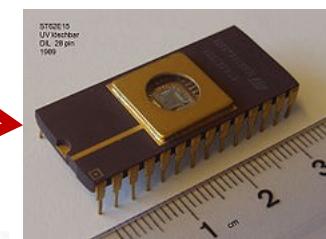
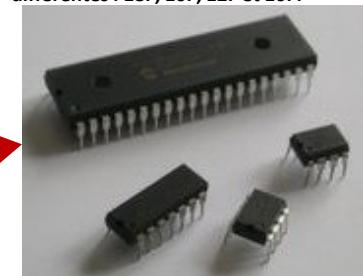
# FAMILLES DE MICRO-CONTROLEUR

- la famille Atmel AT91 ;
- la famille Atmel AVR ; →
- le C167 de Siemens/Infineon ;
- la famille Hitachi H8 ;
- la famille Intel 8051, qui ne cesse de grandir ; de plus, certains processeurs récents utilisent un cœur 8051, qui est complété par divers périphériques (ports d'E/S, compteurs/temporiseurs, convertisseurs A/N et N/A, chien de garde, superviseur de tension, etc.) ;
- l'Intel 8085, à l'origine conçu pour être un microprocesseur, a en pratique souvent été utilisé en tant que microcontrôleur ;
- La famille Motorola/Freescale 68HC08 , 68HC12 et 68HC11: lecteurs de codes-barres, programmeurs de cartes d'hôtel, robots d'amateurs, et d'autres systèmes embarqués.
- la famille des PIC de Microchip ; →
- la famille des ST6 de STMicroelectronics ; →
- la famille ADuC d'Analog Devices ;
- la famille PICBASIC de Comfile Technology;
- la famille MSP430 de Texas Instruments.
- la famille 8080, z80, Rabbit: le 8080 est un des grands ancêtres, mais z80 et Rabbit sont encore bien vivants →
- la famille PSoC de Cypress
- la famille LPC21xx ARM7-TDMI de Philips
- la famille NEC: V800, K0...

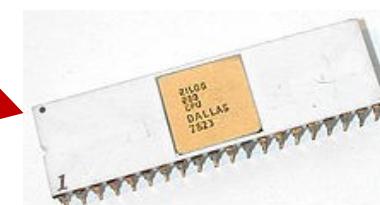


Le MC68HC11A8 est disponible en version DIP à 48 broches, ainsi que PLCC à 52 broches, comme ici.

Quatre microcontrôleurs PIC de familles différentes : 18F, 16F, 12F et 10F.



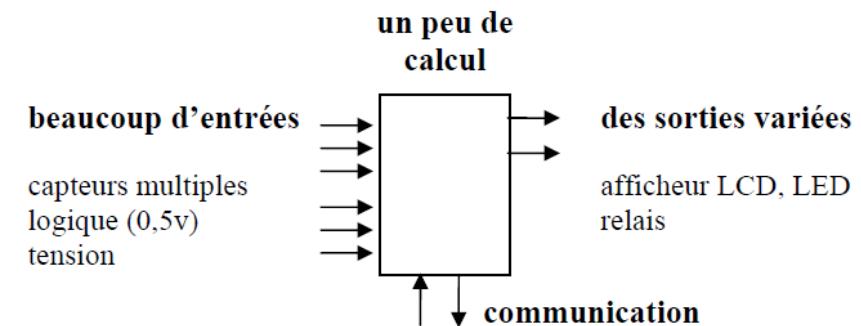
Microcontrôleur ST6 équipé d'une mémoire EPROM effaçable aux UV.



Ce processeur fut commercialisé pour la toute première fois en juillet 1976. Au début des années 1980 il fut très populaire dans la conception des ordinateurs 8-bits comme le Radio Shack TRS-80, les Sinclair ZX80, ZX81, ZX Spectrum, le standard MSX, les Amstrad CPC et plus tard dans les systèmes embarqués.

Nous ne pourrons pas répondre entièrement à cette question, tellement il y aurait à dire.  
Mais voici une explication simple, réductrice, mais qui vous donne un aperçu de ses fonctions :

- 1 - Un microcontrôleur prend quelque chose en entrée
- 2 - Il prend une décision par rapport à un programme que vous avez entré dans sa mémoire, il calcule, il communique...
- 3 – Il produit un résultat en sortie



Les plateformes actuelles basées sur des microcontrôleurs (comme l'Arduino) sont de véritables petits ordinateurs prenant des tas d'entrées possibles, dialoguant entre eux par WIFI, affichant des éléments sur des écrans LCD, actionnant des moteurs etc.

Les programmes peuvent être relativement complexes... Mais ce n'est pas le but en réalité !  
Même si les microcontrôleurs sont relativement puissants et qu'on peut les travestir en ordinateur, le but est normalement d'être des interfaces autonomes, agrégeant les capteurs et actionneurs et qui déléguent aux ordinateurs plus complexes les tâches de calculs.

Retenez tout de même que:  
une réalisation logicielle est toujours plus lente qu'une réalisation en logique câblée : le microprocesseur exécute une instruction à la fois (voir aussi le langage VHDL-AMS => programmation FPGA)

Les microcontrôleurs = avantage des microprocesseurs mais limités aux applications ne nécessitant pas trop de puissance de calcul, nombre de composants très réduit, mais souvent surdimensionnement devant les besoins de l'application

#### => Les avantages des microcontrôleurs :

- Diminution de l'encombrement du matériel et du circuit imprimé
- Simplification du tracé du circuit imprimé (plus besoin de tracer de bus !)
- Augmentation de la fiabilité du système
  - nombre de composants ↓
  - connexions composants/supports et composant circuit imprimé ↓
- Intégration en technologie MOS, CMOS, ou HCMOS (High Speed CMOS)
  - diminution de la consommation
- Le microcontrôleur contribue à réduire les coûts à plusieurs niveaux:
  - moins cher que les composants qu'il remplace
  - Diminution des coûts de main d'œuvre (conception et montage)
- Environnements de programmation et de simulation évolués

#### => Les défauts des microcontrôleurs :

- le microcontrôleur est souvent surdimensionné devant les besoins de l'application
- Investissement dans les outils de développement adaptés (sauf pour l'Arduino...)
- Écrire les programmes, les tester et tester leur mise en place sur le matériel qui entoure le microcontrôleur
- Incompatibilité possible des outils de développement pour des microcontrôleurs de même marque

# QUE FAIRE AVEC TOUT CELA ?

## Exemple:

- Vous souhaitez mesurer en temps réel la température et l'humidité d'un serre et déclencher un arrosage. Vous pouvez aussi confier au microcontrôleur la tâche d'envoyer périodiquement par WIFI les données relevées à un ordinateur et qui commandera en retour au microcontrôleur des arrosages plus « intelligents ».
- Analyser un signal vidéo et faire de la détection de mouvement, c'est déjà limite pour un microcontrôleur (même si certains le peuvent – comme l'Arduino par exemple)

*Mais faire une analyse des visages avec reconnaissance faciale vous demanderait d'étoffer sérieusement le kit de développement de votre microcontrôleur !*

## Applications:

- Les microcontrôleurs sont souvent utilisés dans l'**élaboration de systèmes embarqués**, nécessitant des traitements spécialisés (autoradios, téléphones portables, lecteur mp3, GPS, etc.)
- Mais aussi pour réaliser du prototypage rapide d'applications.

*Ces circuits intégrés sont également très prisés en robotique amateur et permettent de réaliser de nombreuses applications, y compris des robots autonomes, les automatismes en modélisme (maquettes de réseau ferroviaire...).*

## Que faire avec tout ça ?

Nous allons déjà nous amuser avec une carte intégrant un composant ATMEL AVR sur la plaquette de prototypage rapide Arduino, ce qui vous délestera de la tâche difficile de comprendre la structure intime d'un tel microcontrôleur.

## II - LA PLATEFORME ARDUINO®

# QU'EST-CE QU'UN ARDUINO ?

## Un hardware



ADXL3xx | Arduino 0022

```

File Edit Sketch Tools Help
ADXL3xx
ADXL3xx

se constants describe the pins. They won't change:
int groundpin = 18;           // analog input pin 4 -- ground
int powerpin = 19;            // analog input pin 5 -- voltage
int xpin = A3;                // x-axis of the accelerometer
int ypin = A2;                // y-axis
int zpin = A1;                // z-axis (only on 3-axis models)

void setup()
{
  // initialize the serial communications:
  Serial.begin(9600);
}

void loop()
{
  // provide ground and power by using the analog inputs as normal
  // digital pins. This makes it possible to directly connect the
  // breakout board to the Arduino. If you use the normal 5V and
  // GND pins on the Arduino, you can remove these lines.
  pinMode(groundpin, OUTPUT);
  pinMode(powerpin, OUTPUT);
  digitalWrite(groundpin, LOW);
  digitalWrite(powerpin, HIGH);

  // print the sensor values:
  Serial.print(analogRead(xpin));
  Serial.print("\t");
  Serial.print(analogRead(ypin));
  Serial.print("\t");
  Serial.print(analogRead(zpin));
  Serial.println();
  // delay before next reading:
  delay(100);
}

Done compiling.

Binary sketch size: 2628 bytes (of a 32256 byte maximum)
1

```

Une communauté

<http://arduino.cc>


**ARDUINO PLAYGROUND**

The playground is a publicly-editable wiki about Arduino.

- Manuals and Curriculum
- Board Setup and Configuration
- Development Tools
- Interfacing With Hardware
  - Output
  - Input


**ARDUINO FORUM**

Using Arduino

- Installation & Troubleshooting
 

For problems with Arduino itself, NOT your project  
Last post: Re: Unable to upload to ... by openpolis66 on Today at 10:08:04 AM
- Project Guidance
 

Advice on general approaches or feasibility  
Last post: Re: How to control ph by paulus on Today at 10:30:04 AM
- Programming Questions
 

Understanding the language, error messages, etc.  
Last post: Re: a pointer to a multi... by kasperkamperman.com on Today at 10:20:00 AM
- General Electronics
 

Resistors, capacitors, breadboards, soldering, etc.  
Last post: Re: Isolated digipot? by Gravymad on Today at 10:23:58 AM
- Microcontrollers
 

Standalone or alternative microcontrollers, in-system programming, bootloaders, etc.  
Last post: Re: ATtiny85 revision C by Coding\_Buddy on Today at 10:18:01 AM
- LEDs and Multiplexing
 

Controlling lots of inputs and outputs  
Last post: Re: 5940nt flickering by lbourdon on Today at 10:40:01 AM
- Displays
 

LCDs, OLEDs, PAL, NTSC, etc.  
Last post: Re: My 20x2 LCD doesn't w... by Dorus on Today at 10:37:58 AM
- Audio
 

Sound processing and generation, using WAV and MP3 players, using MIDI  
Last post: Re: Microphone to detect... by AWOL on Today at 08:18:43 AM
- Motors, Mechanics, and Power
 

Controlling big heavy things that move, need high voltages, high current, or both  
Last post: Re: AccelStepper help by Mark7 on Today at 08:51:02 AM
- Sensors
 

sensors, analog inputs, filtering, etc.  
Last post: Re: Ultrasonic sensor range by jay on Today at 08:45:25 AM

The Arduino Playground

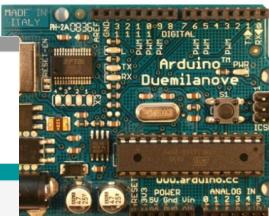


Photo of an Arduino Board

Arduino Playground, a wiki where all the users of Arduino and benefit from their collective research.

to post and share your own code, circuit diagrams, tutorials, ns, tips and tricks, and after all the hard work, to show off your ne can edit and add to the pages here.

round is a **work in progress**. We can use all the help you can

Le microcontrôleur de la carte est programmé en utilisant le langage de programmation Arduino (basé sur **WIRING**) et l'environnement de développement Arduino (basé sur le **PROCESSING**).

Il y a de nombreux microcontrôleurs et de nombreuses plateformes basées sur des microcontrôleurs disponibles pour l'électronique programmée: Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, et beaucoup d'autres qui offrent des fonctionnalités comparables.

## Alors pourquoi Arduino ?

- **Un environnement de programmation clair et simple:** L'environnement de programmation Arduino (= le logiciel Arduino) est facile à utiliser pour les débutants, tout en étant assez flexible pour que les utilisateurs avancés puissent en tirer profit également;
- **Pas cher :** les cartes Arduino sont relativement peu coûteuses comparativement aux autres plateformes. La moins chère des versions du module Arduino peut être assemblée à la main, et même les cartes Arduino pré-assemblées coûtent moins de 25€ (microcontrôleur inclus...) !!!
- **Multi-plateforme :** Le logiciel Arduino, écrit en Java, tourne sous les systèmes d'exploitation Windows, Macintosh et Linux. La plupart des systèmes à microcontrôleurs sont limités à Windows.
- **Plateforme de communication importante:**
  - complètement « stand-alone », ou
  - il parle à d'autres dispositifs: 'C', Flash, Processing (Java), Pure data, MAX/MSP, Ruby, Python, .NET...
- **Sur l'internet, on trouve :**
  - Une communauté d'utilisateurs.
  - Des guides d'utilisation.
  - Des exemples.
  - Des forums d'entraide...

## 1 - Logiciel Open Source et extensible: => on peut l'utiliser et le modifier librement.

Le logiciel Arduino et le langage Arduino sont publiés sous licence **open source**, disponible pour être complétés par des programmeurs expérimentés.

Le langage peut être aussi étendu à l'aide de bibliothèques C++, et les personnes qui veulent comprendre les détails techniques peuvent reconstruire le passage du langage Arduino au langage C pour microcontrôleur AVR sur lequel il est basé.

*De la même façon, vous pouvez ajouter du code du langage AVR-C directement dans vos programmes Arduino si vous voulez.*

## 2 - Matériel Open source et extensible: => on peut le copier, le fabriquer et le modifier librement.

Les cartes Arduino sont basées sur les microcontrôleurs Atmel ATMEGA8, 168, 328, etc...

Dont les schémas des modules sont publiés sous une licence Creative Commons (CC).

Les concepteurs de circuits expérimentés ou relativement inexpérimentés peuvent réaliser leur propre version des cartes Arduino, en les complétant et en les améliorant.

**Mais attention: un logiciel ou un matériel OPEN SOURCE est avant tout un logiciel ou un matériel dont les codes ou les plans sont accessibles et modifiables par tous.**

Ce partage de la connaissance n'empêche pas la rémunération !

*La société Arduino est une société commerciale, de nombreuses entreprises travaillent et prospèrent autour du libre (~100M\$ au total en 2010)*

mais permet surtout des échanges et une circulation de la connaissance...

*A une période où les interrogations quant au partage de la connaissance sont grandes, avoir une petite réflexion éthique sur la marchandisation de la connaissance et de son travail me semble salutaire...*

La plateforme Arduino est basée sur l'architecture des microcontrôleurs Atmel ATMEGA qui utilisent un cœur AVR (Advanced Virtual RISC):



AVR est le terme utilisé par Atmel pour désigner le cœur du processeur et la famille de microcontrôleurs RISC les implémentant (développé en 1996 par ATMEL corporation).

L'architecture a été conçue par deux étudiants de l'institut de technologie de Norvège (NTH : Norges Tekniske Høgskole): Alf-Egil Bogen and Vegard Wollan

=> Alf-Egil Bogen and Vegard Wollan RISC microcontroller qui donne aussi Advanced Virtual RISC.

#### Historiquement:

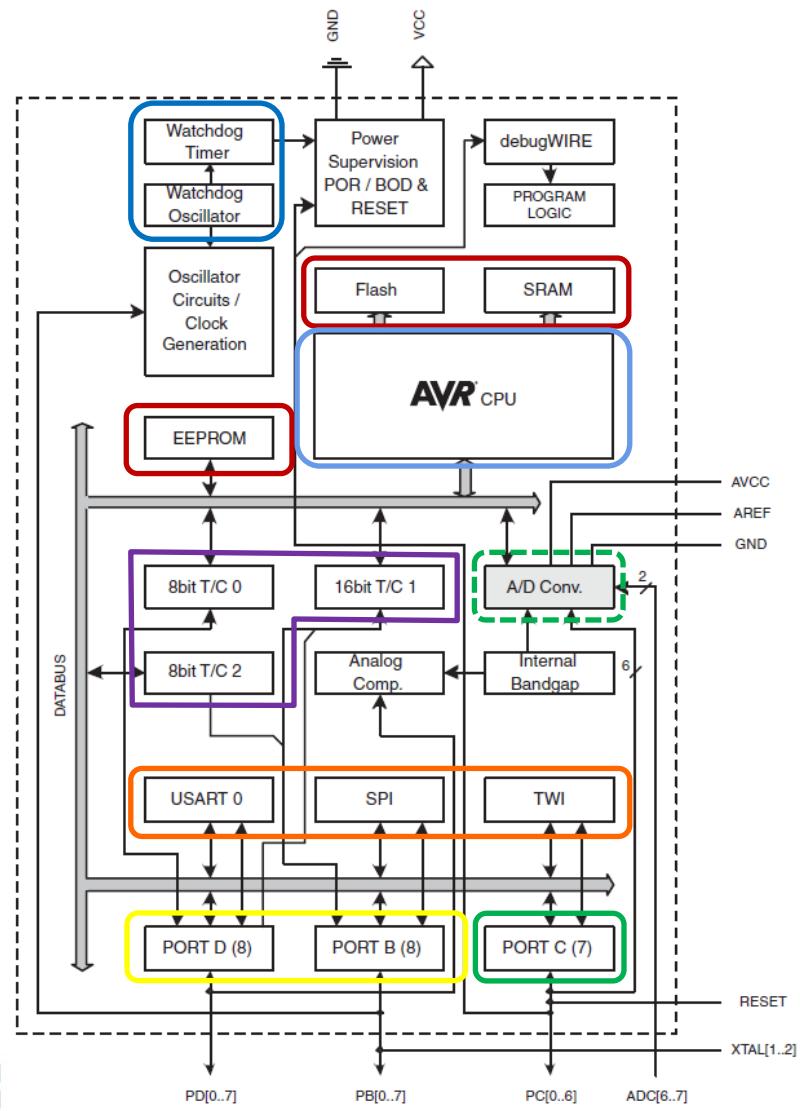
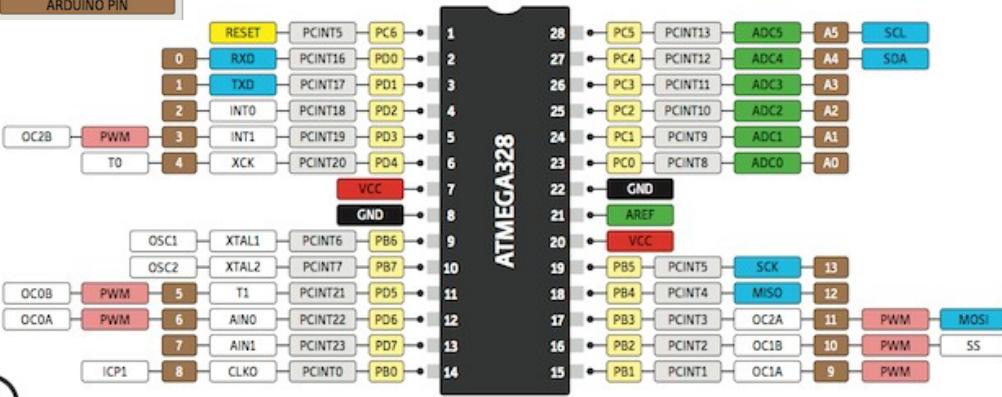
l'AT90S8515 a été le premier microcontrôleur basé sur l'architecture AVR MAIS c'est l'AT90S1200 qui fut le 1er microcontrôleur à rencontrer un succès commercial dans l'année 1997.



**Attention: le langage machine des AVR est incompatible avec la famille des PIC de Microchip !**

# ARCHITECTURE ATMEL AVR

(PCINT14/RESET) PC6	1	28	□ PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	□ PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	□ PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	□ PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	□ PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	□ PC0 (ADC0/PCINT8)
VCC	7	22	□ GND
GND	8	21	□ AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	□ AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	□ PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	□ PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	□ PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	□ PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	□ PB1 (OC1A/PCINT1)



SOURCE: [http://www.atmel.com/dyn/resources/prod\\_documents/8271S.pdf](http://www.atmel.com/dyn/resources/prod_documents/8271S.pdf)

**Le cœur du noyau est basé sur une architecture RISC de 131 instructions.**  
**Ce nombre d'instructions est assez élevé pour une telle architecture => Advanced RISC Architecture**

**Jusqu'à 20 Millions d'instructions par secondes (MIPS) pour un Quartz de 20 MHz (la majorité des instructions étant réalisée en 1 ou 2 cycles)**  
**=> les microcontrôleurs AVR sont donc très rapide !**

L'architecture interne de l'ATMEGA328 nous permet de remarquer les périphériques suivants :

**2 Timer/Compteur 8 bits avec facteur de pré-division indépendant**

**1 Timer/Compteur 16 bits avec facteur de pré-division indépendant**

**1 Horloge Temps réel avec quartz externe**

**6 canaux PWM**

**1 Convertisseur Analogique/Numérique 8 voies (résolution de 10 bits)**

**1 Interface de communication Asynchrone USART**

**1 interface série Synchrone I<sup>2</sup>C (Philips I<sup>2</sup>C Compatible)**

**1 Interface de communication Synchrone SPI (servant aussi à la programmation In-situ)**

**2 interruptions sur les changements d'états de pins**

**1 Compteur Watchdog programmable**

...

La liste ci-dessus énumère la liste des périphériques qui ont un rôle majeur dans la communication avec le monde extérieur au microcontrôleur.

**Il faut compter aussi sur d'autres fonctions typiquement internes, comme les différents type de mémoires Flash...**

## Features

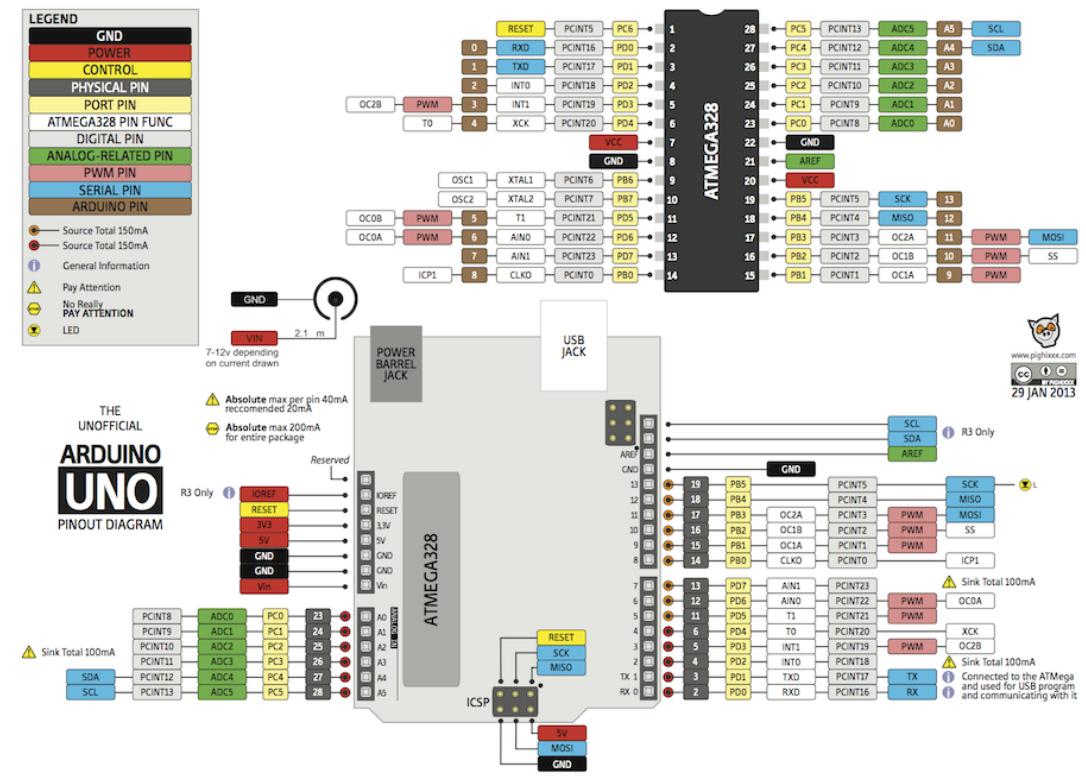
- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 4/8/16/32KBytes of In-System Self-Programmable Flash program memory
  - 256/512/512/1KBytes EEPROM
  - 512/1K/1K/2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C<sup>(4)</sup>
  - Optional Boot Code Section with Independent Lock Bits
  - In-System Programming by On-chip Boot Program
  - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix® acquisition
  - Up to 64 sense channels
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
  - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
  - Temperature Measurement
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Byte-oriented 2-wire Serial Interface (Philips I<sup>2</sup>C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 85°C
- Speed Grade:
  - 0 - 4MHz@1.8 - 5.5V, 0 - 10MHz@2.7 - 5.5V, 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

# LA CARTE ARDUINO « UNO »

La carte Arduino « Uno » est une carte à microcontrôleur basée sur l'ATmega328 qui dispose:

- de 14 broches numériques d'entrées/sorties (dont 6 sorties PWM (largeur d'impulsion modulée)),
- de 6 entrées analogiques (qui peuvent également être utilisées en broches entrées/sorties numériques),
- d'un quartz 16MHz,
- d'une connexion USB,
- d'un connecteur d'alimentation jack,
- d'un connecteur ICSP (programmation "in-circuit"), cf. <http://arduino.cc/en/Hacking/Programmer>
- et d'un bouton de réinitialisation (reset).

Elle contient tout ce qui est nécessaire pour le fonctionnement du microcontrôleur; Pour l'utiliser, il suffit de la connecter à un PC par l'USB (ou avec un adaptateur secteur ou une pile).



Microcontrôleur	ATmega328
Tension de fonctionnement	5V
Tension d'alimentation (recommandée)	7-12V
Tension d'alimentation (limites)	6-20V
Broches E/S numériques	14 (dont 6 disposent d'une sortie PWM)
Broches d'entrées analogiques	6 (utilisables en broches E/S numériques)
Intensité maxi disponible par broche E/S (5V)	40 mA ( <b>ATTENTION : 200mA cumulé pour l'ensemble des broches E/S</b> )
Intensité maxi disponible pour la sortie 3.3V	50 mA
Intensité maxi disponible pour la sortie 5V	Fonction de l'alim utilisée - 500 mA max si port USB utilisé seul
Mémoire Programme Flash	32 KB (ATmega328) dont 0.5 KB sont utilisés par le bootloader
Mémoire SRAM (mémoire volatile)	2 KB (ATmega328)
Mémoire EEPROM (mémoire non volatile)	1 KB (ATmega328)
Vitesse d'horloge	16 MHz

### Entrées/Sorties Analogiques:

Les cartes Arduino disposent de **6 entrées analogiques (A<sub>0</sub> à A<sub>5</sub>)**, chacune pouvant fournir une mesure de résolution 10 bits (i.e. sur 1024 niveaux) à l'aide de la fonction [analogRead\(\)](#) du langage Arduino.

Par défaut, ces broches mesurent entre le 0V (valeur 0) et le 5V (valeur 1023), mais il est possible de modifier la référence supérieure de la plage de mesure en utilisant la broche AREF et l'instruction [analogReference\(\)](#) du langage Arduino.

**Broches analogiques spécifiques:** A4 (SDA) et A5 (SCL). Supportent les communications de protocole I<sup>2</sup>C (ou interface TWI (Two Wire Interface - Interface "2 fils")), disponible en utilisant [la librairie Wire/I<sup>2</sup>C](#).

*Note : les broches analogiques peuvent aussi être utilisées en tant que broches numériques: elles sont dans ce cas numérotées en tant que broches numériques de 14 à 19.*

## ENTRÉES ET SORTIES NUMÉRIQUES :

Les 14 broches numériques de la carte UNO (D0 à D13) sont utilisées en 5V soit comme une entrée numérique, soit comme une sortie numérique, en utilisant les instructions [pinMode\(\)](#), [digitalWrite\(\)](#) et [digitalRead\(\)](#) du langage Arduino.

Chaque broche peut fournir ou recevoir un maximum de 40mA d'intensité et dispose d'une résistance interne de "rappel au plus" (pull-up) (déconnectée par défaut) de 20-50 KOhms.

Cette résistance interne s'active sur une broche en entrée à l'aide de l'instruction [digitalWrite\(broche, HIGH\)](#).

## De plus, certaines broches ont des fonctions spécialisées :

• **Communication Série:** Broches D0 (RX) et D1 (TX). Utilisées pour recevoir (RX) et transmettre (TX) les données séries de niveau TTL.

• **Interruptions Externes:** Broches D2 et D3. Ces broches peuvent être configurées pour déclencher une interruption sur une valeur basse, sur un front montant ou descendant, ou sur un changement de valeur, voir [attachInterrupt\(\)](#)...

• **Impulsion PWM (largeur d'impulsion modulée):** Broches 3, 5, 6, 9, 10, et 11. Fournissent une impulsion PWM 8-bits à l'aide de l'instruction [analogWrite\(\)](#).

• **SPI (Interface Série Périphérique):** Broches D10 (SS), D11 (MOSI), D12 (MISO), D13 (SCK). Supportent la communication SPI (Interface Série Périphérique) disponible avec la [librairie pour communication SPI](#). Les broches SPI sont également connectées sur le connecteur ICSP.

• **LED:** Broche D13. Une LED inclue dans la carte est connectée à la broche 13.

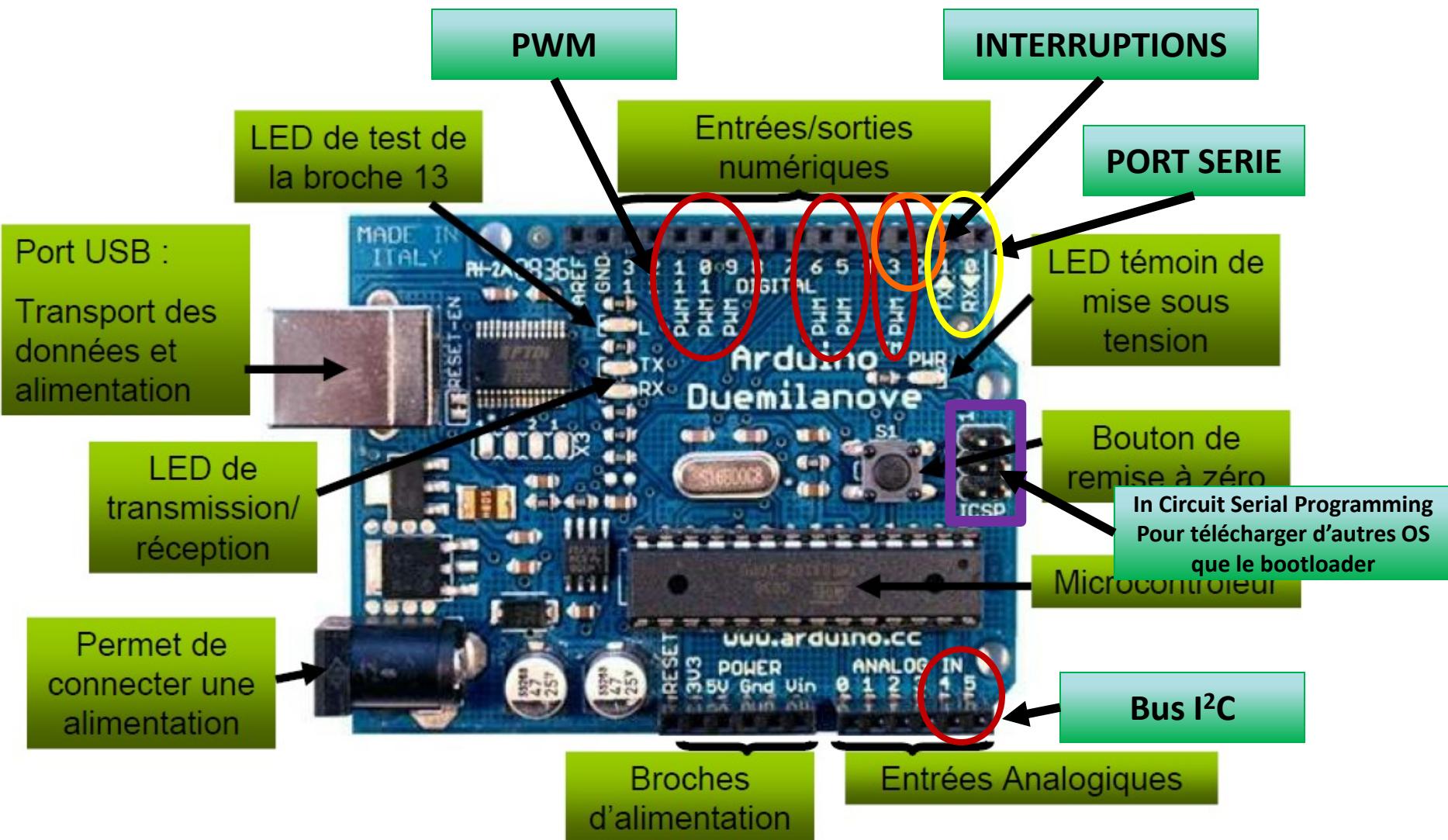
Lorsque la broche est au niveau HAUT, la LED est allumée, lorsque la broche est au niveau BAS, la LED est éteinte.

## Autres broches:

• **AREF :** Tension de référence pour les entrées analogiques (si différent du 5V). Utilisée avec l'instruction [analogReference\(\)](#).

• **RESET :** Mettre cette broche au niveau BAS entraîne la réinitialisation du microcontrôleur. Typiquement, cette broche est utilisée pour ajouter un bouton de réinitialisation sur le circuit qui bloque celui présent sur la carte.

# AUTOPSIE DE LA CARTE ARDUINO



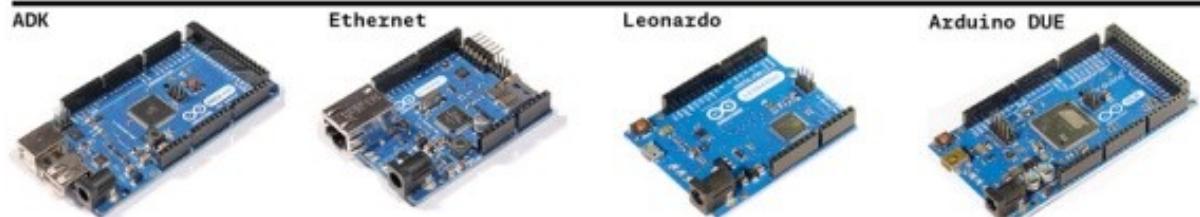
# FAMILLE ARDUINO & QUELLES APPLICATIONS

Arduino	Micro-contrôleur	Flash ko	EEPROM ko	SRAM ko	Broches d'E/S numériques	...avec PWM	Broches d'entrée analogique	Type d'interface USB	Dimensions pouces	Dimensions mm
Diecimila	ATmega168	16	0,5	1	14	6	6	FTDI	2,7" x 2,1"	68,6 mm x 53,3 mm
Duemilanove	ATmega168/328P	16/32	0,5/1	1/2	14	6	6	FTDI	2,7" x 2,1"	68,6 mm x 53,3 mm
Uno	ATmega328P	32	1	2	14	6	6	ATmega16U2	2,7" x 2,1"	68,6 mm x 53,3 mm
Leonardo	ATmega32U4	32	1	2,5	20	7	12	ATmega32U4	2,7" x 2,1"	68,6 mm x 53,3 mm
Mega	ATmega1280	128	4	8	54	15	16	FTDI	4" x 2,1"	101,6 mm x 53,3 mm
Mega2560	ATmega2560	256	4	8	54	15	16	ATmega8U2	4" x 2,1"	101,6 mm x 53,3 mm
Due	Atmel SAM3X8E	512	0	96	54	12	12	SAM3X8E (USB Host), ATmega16u2 (programmation)	4" x 2,1"	101,6 mm x 53,3 mm
Fio	ATmega328P	32	1	2	14	6	8	Aucune	1,6" x 1,1"	40,6 mm x 27,9 mm
Nano	ATmega168 or ATmega328	16/32	0,5/1	1/2	14	6	8	FTDI	1,70" x 0,73"	43 mm x 18 mm
LilyPad	ATmega168V or ATmega328V	16	0,5	1	14	6	6	Aucune	2" ø	50 mm ø
Yun <sup>12</sup>	ATmega32u4	32	1	2,5	20	7	12			73 mm x 53 mm
Esplora	ATmega32U4	32	1	2,5	N/A	N/A	N/A	ATmega32U4	6,5" x 2,4"	165,1 mm x 60,96 mm

Source: <http://fr.wikipedia.org/wiki/Arduino>

# FAMILLE ARDUINO...

Les nouveautés...



Microcontroller	ATmega2560	ATmega328	ATmega32U4	Atmel SAM3U4E ARM Cortex M3
Clock	16 MHz	16 MHz	16 MHz	96 MHz
Flash Memory	256 KB	32 KB	32 KB	256 KB
SRAM	8 KB	2 KB	3.3 KB	50 KB
Digital I/O Pins	54	14 (10)	14	54
Analog Pins	16	6	6	16 (12bit)
	Android ADK Compatible USB Host Develop your own android accessory!	Wiznet W5100 Ethernet interface Optional PoE Module Bring your project online!	Onboard USB controller Build your own USB devices!	Onboard dual-channel DAC Bringing 32 bit power to Arduino!
Wifi		TinkerKit		Arduino Robot System 
	Avr32 co-processor with fully open-source firmware H&D wifi module Easy to upgrade firmware Fully Hackable!	Breadboard-free electronic prototyping 30+ different modules Easy to use instructions & tutorials!	Arduino based dual platform robot Multiprocessor TinkerKit-compatible Program your own behaviours!	Look for us at Maker Faire 2011/New York 

[www.arduino.cc](http://www.arduino.cc)

Source: <http://fr.wikipedia.org/wiki/Arduino>



Arduino Uno



Arduino Leonardo



Arduino GSM Shield



Arduino Due



Arduino Yún



Arduino Ethernet  
Shield



Arduino Tre



Arduino Micro



Arduino WiFi Shield



Arduino Robot



Arduino Esplora



Arduino Wireless SD

ENTRY LEVEL	ARDUINO/GENUINO UNO	ARDUINO PRO	ARDUINO PRO MINI	ARDUINO/GENUINO MICRO
	ARDUINO NANO	ARDUINO/GENUINO STARTER KIT	ARDUINO BASIC KIT	
	ARDUINO MOTOR SHIELD			
ENHANCED FEATURES	ARDUINO/GENUINO MEGA	ARDUINO ZERO	ARDUINO DUE	ARDUINO PROTO SHIELD
INTERNET OF THINGS	ARDUINO YÚN	ARDUINO ETHERNET SHIELD	ARDUINO GSM SHIELD	ARDUINO WIFI SHIELD 101
WEARABLE	ARDUINO GEMMA	LILYPAD ARDUINO USB	LILYPAD ARDUINO MAIN BOARD	
	LILYPAD ARDUINO SIMPLE	LILYPAD ARDUINO SIMPLE SNAP		
3D PRINTING	MATERIA 101			

BOARDS  
  MODULES  
  SHIELDS  
  KITS  
  ACCESSORIES  
  COMING NEXT

## II – 2 L'ENVIRONNEMENT DE DEVELOPPEMENT

Blink | Arduino 0018

File Edit Sketch Tools Help

Blink

```

int ledPin = 13; // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts

void setup()
{
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}

// the loop() method runs over and over again,
// as long as the Arduino has power

void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(1000);
  digitalWrite(ledPin, LOW); // set the LED off
  delay(1000);
}

```

Done compiling.

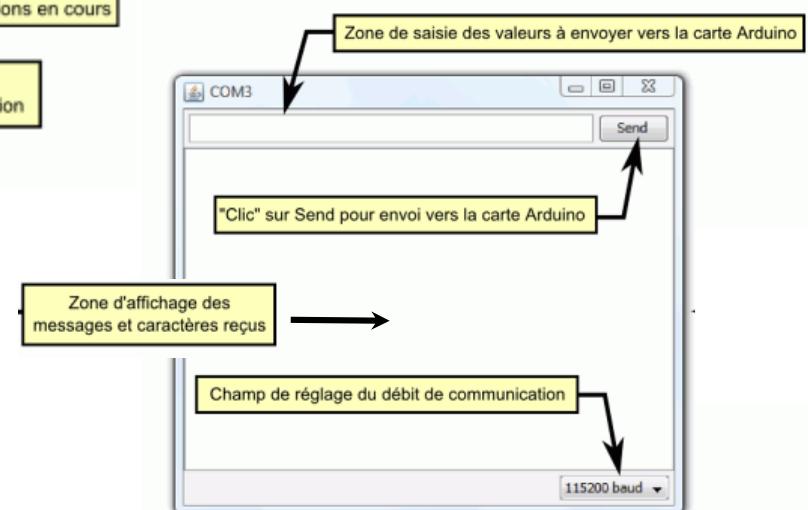
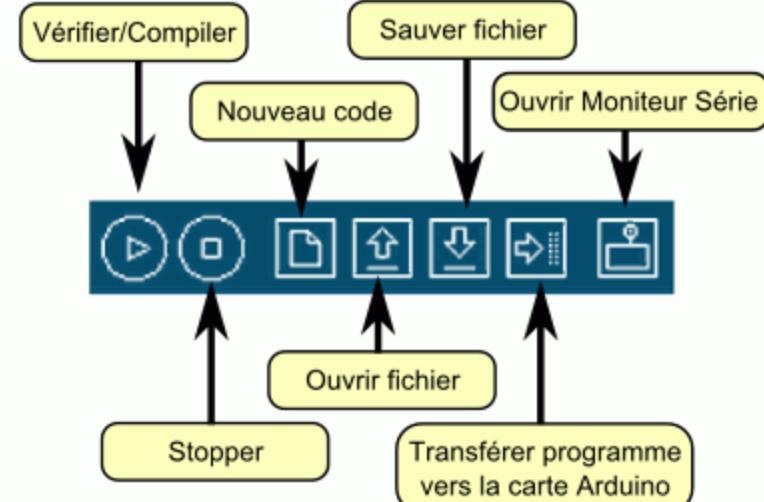
Binary sketch size: 896 bytes (of a 30720 byte maximum)

Barre de Menu  
Barre de Boutons  
Onglets des fichiers ouverts

Fenêtre d'édition des programmes

Zone de messages des actions en cours

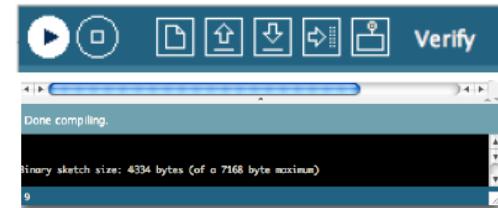
Console d'affichage des messages de compilation



L'Arduino se programme dans un langage évolué ( $\neq$  langage machine) mélangeant du C et du C++ restreint et adapté aux possibilités de la carte (langages utilisés par les professionnels).

## 1. On conçoit ou on ouvre un programme existant avec le logiciel ARDUINO

## 2. On vérifie ce programme avec le logiciel ARDUINO (compilation).



## 3. Si des erreurs sont signalées, on modifie/corrigé le programme.

## 4. On charge (upload) le programme sur la carte,

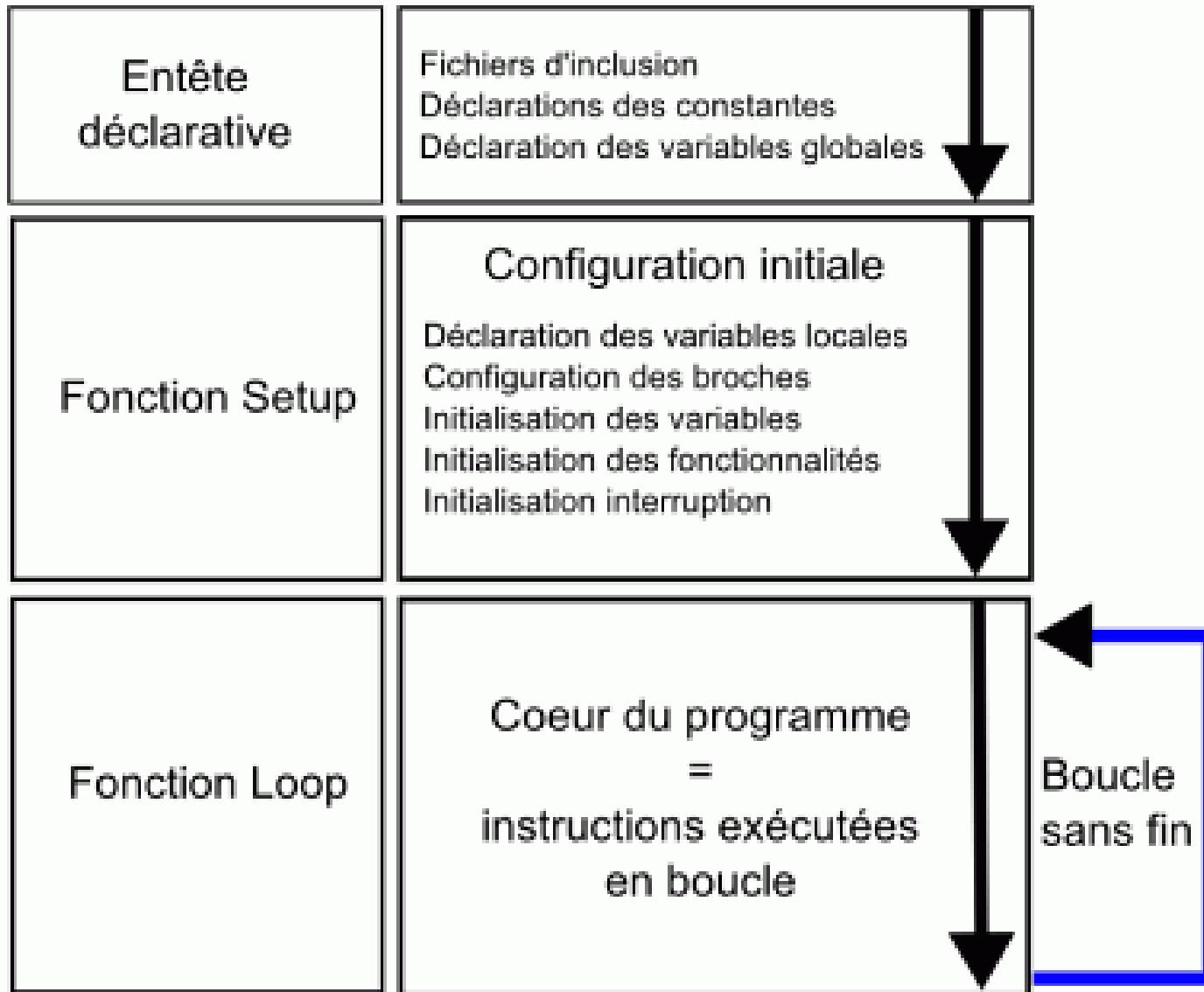


## 5. On câble le montage électronique.

## 6. L' exécution du programme est automatique après quelques secondes si l'arduino est branché (éventuellement on fait un reset)

## 7. On alimente la carte soit par le port USB, soit par une source d'alimentation autonome (pile 9 volts par exemple)

## 8. On vérifie que le montage fonctionne (et éventuellement un déboggage)



**Commentaires**

Toujours écrire des commentaires sur le programme: soit en multiligne, en écrivant entre des /\* \*/, soit sur une ligne de code en se séparant du code avec //

(Syntaxe en marron, paramètres utilisateur en vert)

```
/* Ce programme fait clignoter une LED branchée sur la broche 13
 * et fait également clignoter la diode de test de la carte
 */
```

**Définition des variables:**

Pour notre montage, on va utiliser une sortie numérique de la carte, qui est par exemple la 13 ème sortie numérique. Cette variable doit être définie et nommée ici: on lui donne un nom arbitraire **BrocheLED**. Le mot de la syntaxe est pour désigner un nombre entier est **int**

```
int BrocheLED = 13; // Définition de la valeur 13 et du nom de la broche à utiliser
```

**Configuration des entrées-sorties void setup():**

Les broches numériques de l'Arduino peuvent aussi bien être configurées en entrées numériques ou en sorties numériques. Ici on va configurer **BrocheLED** en sortie. **pinMode ( nom, état)** est une des quatre fonctions relatives aux entrées-sorties numériques.

```
void setup()
{
    pinMode(BrocheLED, OUTPUT); // configure BrocheLED comme une sortie
}
```

**Programmation des interactions void loop():**

Dans cette boucle, on définit les opérations à effectuer, dans l'ordre:

- **digitalWrite ( nom, état)** est une autre des quatre fonctions relatives aux entrées-sorties numériques.
- **delay(temps en millisecondes)** est la commande d'attente entre deux autres instruction
- Chaque ligne d'instruction est terminée par un point virgule
- Ne pas oublier les accolades, qui encadrent la boucle.

```
void loop()
{
    digitalWrite(BrocheLED, HIGH); // met la sortie num. à l'état haut (led allumée)
    delay(3000); // attente de 3 secondes
    digitalWrite(BrocheLED, LOW); // met la sortie num. à l'état bas (led éteinte)
    delay(1000); // attente de 1 seconde
}
```

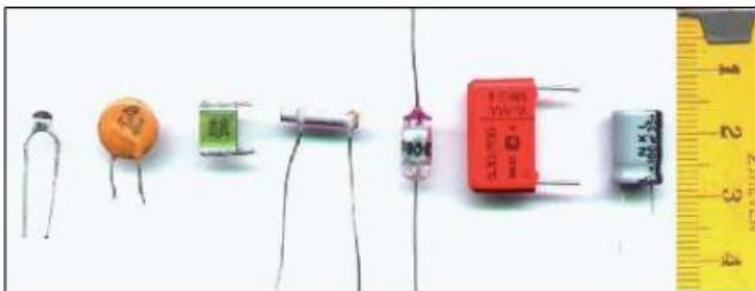
## II – 3 QUELS COMPOSANTS ADRESSABLES ?



### La résistance

La résistance s'oppose au passage du courant, proportionnellement à sa "résistance" exprimée en Ohm. Un code de couleurs, ci dessous permet de reconnaître cette valeur.

Symbol européen



### Les condensateurs

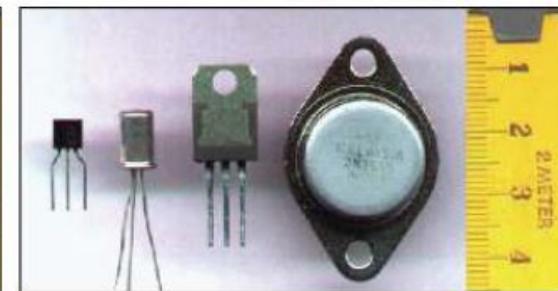
Les condensateurs peuvent stocker un peu de courant si on les charge, mais comme un tonneau percé, ils renvoient ce courant instantanément si ils sont branchés à un organe consommateur de courant.



Reconnaissance de leur valeur:

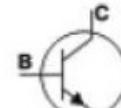
A cause de la diversité des modèles, se reporter aux ressources sur le web

Symbol

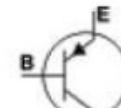


### Le transistor

Le transistor est généralement utilisé comme une sorte de multiplicateur de puissance: lorsqu'on lui fait passer un courant faible, mais variable dans un de ses 3 pattes, il autorise proportionnellement le passage d'un gros courant dans une autre des 3 pattes.



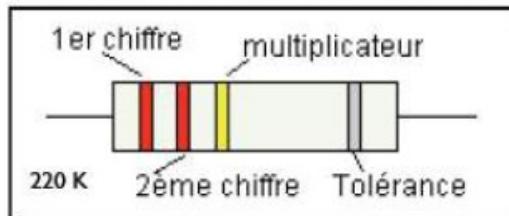
transistor NPN



transistor PNP

Symboles

Code des couleurs des résistances      au delà de 1000 Ohms, on parle en KiloOhms, par exemple 10 K est 10 KiloOhms, puis en MegaOhms notés M



Chiffre	0	1	2	3	4	5	6	7	8	9	Or	Argent
Multiplicateur	1	10	$10^2$ 100	$10^3$ 1000	$10^4$ 10000	$10^5$	$10^6$					
Précision	20%									5%	10%	



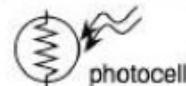
L'interrupteur



L'interrupteur ouvre ou ferme un circuit. Il y a toutes sortes d'interrupteurs.

Sur l'Arduino, utiliser un interrupteur pour déclencher un événement nécessite d'utiliser un composant supplémentaire: une résistance de 10K ohms. Voir "Montages d'électronique interactive".

La cellule photo-électrique (LDR)



photocell

La cellule photo-électrique (LDR)

C'est une résistance variable, en fonction de la luminosité qu'elle reçoit. Sa résistance diminue quand elle reçoit de la lumière. On s'en sert donc de capteur de luminosité. Non polarisée. Pour lire sa valeur avec une Arduino, il faut également l'associer avec une résistance équivalente à sa résistance maxi ( dans le noir) Voir " Montages d'électronique interactive".

Le piezo



Le transducteur piezo-électrique est un composant réversible: il peut aussi bien être utilisé en capteur de chocs ou de vibrations qu'en actionneur pouvant émettre des sons stridents parfois modulables.

Le servo moteur



Le servo-moteur est un moteur (rotatif) qui peut effectuer des rotations très précises (dans une portion de tour seulement) et en un certain nombre de pas (de micro-déplacements). Il y a toutes sortes de servo moteurs.. Un des avantages des servo moteurs est sa possibilité de maintenir avec force une position donnée. On peut piloter des rotations avec l'Arduino, quelques fois directement avec la carte si le moteur n'est pas trop puissant, sinon en passant par un montage associé.

Le potentiomètre



Le potentiomètre



Le potentiomètre, rotatif comme ici, ou à glissière, est une résistance variable. Entre les extrémités, il y a la résistance maximale. La patte centrale est le curseur. C'est la résistance entre cette patte centrale et une extrémité que l'on peut faire varier en tournant le bouton. Le potentiomètre est donc un capteur. Il se branche sur les entrées analogiques de l'Arduino. De très nombreux capteurs sont basés sur le principe de résistance variable et se câblent presque de la même façon: la cellule photo-électrique, le capteur de pression, le fil résistif, etc

Le relais



Le relais



Le relais est un composant à 4 broches minimum. C'est un interrupteur que l'on peut commander en envoyant un petit courant. Au repos, il est normalement fermé, ou normalement ouvert, selon le modèle. On peut s'en servir avec l'Arduino pour commander des machines en haute tension ( 230V par exemple), ou pour déclencher toute machine ou lumière.

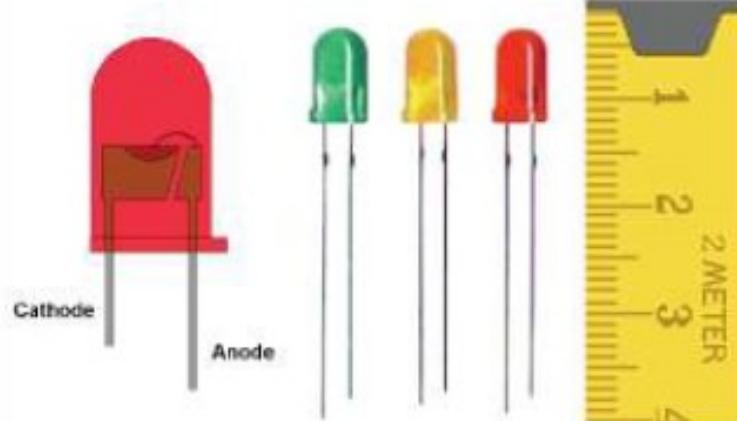
### La diode



### La diode

La diode ne laisse passer le courant que dans un seul sens. C'est un composant polarisé: on reconnaît toujours son anneau coloré d'un coté du composant, correspondant à la cathode.

### La LED



La diode électroluminescente (LED) émet de la lumière. Elle est polarisée: la patte "+" est la plus longue, l'autre patte est la patte "-".

Les broches numériques de l'Arduino, lorsqu'elles sont configurées en sorties et qu'elles sont à l'état 1 ou haut (HIGH), fournissent une tension de 5 volts, supérieure à ce que peut accepter une LED courante, sauf certaines LEDs. Les LED doivent donc être couplées en série avec une résistance.

# Gearhead Motor

## ★ DC Motor with gearbox

- ★ Not fast but provide more torque

## ★ Servo Motor

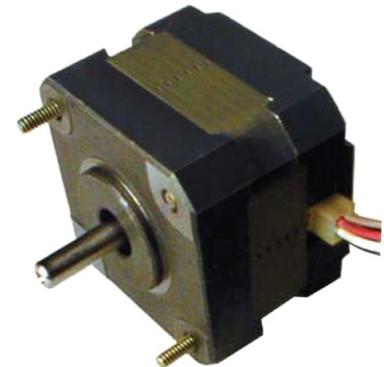
- ★ Gearhead motor with position feedback
- ★ Feedback is often from potentiometer
- ★ Pulsing the motor moves it to particular position within 180 degree range
- ★ Can't move 360 degrees but can be positioned precisely within the 180 degree range



# Stepper Motor

## ★ Precise positioning & 360 degrees range

- ★ Move in discrete steps around a circle
- ★ A 200 step motor would move 1.8 degrees per step around the full 360 degrees
- ★ Continuous rotation in either direction
- ★ Good torque
- ★ Complex to connect



# Solenoids and Actuators

## ★ Linear Motion

- ★ Pull or Push

## ★ Types

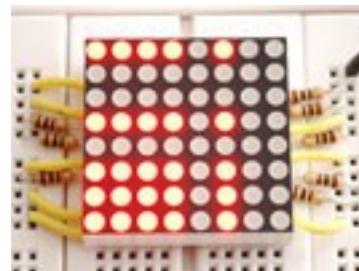
- ★ Solenoid
- ★ Actuator
- ★ Microactuator



# QUELS COMPOSANTS UTILISER AVEC L'ARDUINO ?



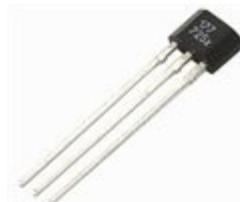
GPS



Matrice de LED



Capteur Force



Capteur Effet Hall



Flex Sensor



Modules Sonar



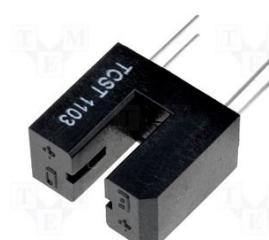
Capteur Inclinaison



Capteur de Gaz



Capteur de mouvement PIR



OptoCoupleur



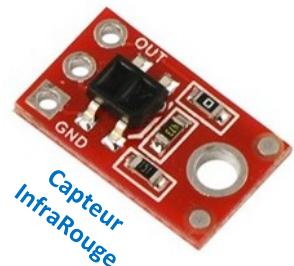
Capteur Inclinaison



Capteur Température/Humidité



Ecran LCD 16\*2



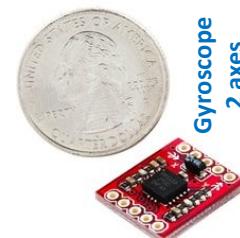
Capteur InfraRouge



Capteur Son



Capteur Flamme



Gyroscope 2 axes



Digital Strip

# QUELS COMPOSANTS UTILISER AVEC L'ARDUINO ?



Capteur Rotation



Mini-capteur Manche



Capteur de chaleur



Capteur Couleur



Camera JPEG



Capteur Lumière



Capteur Lumière



Capteur Vibration



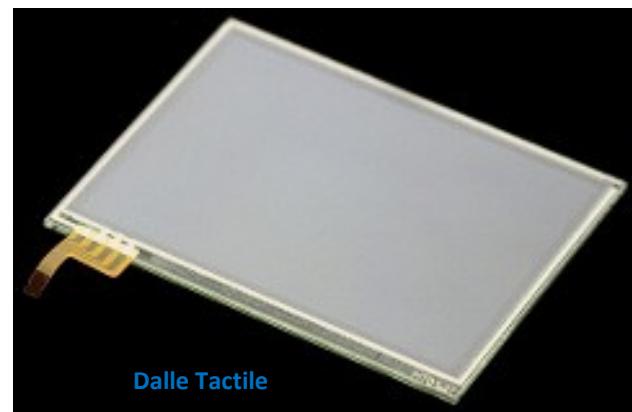
BMP085 Breakout  
Capteur Pression-  
Barométrique



Kinect



Nunchuk



Dalle Tactile

et bien d'autres que nous ne pourrions citer par manque de place...

# CAPTEURS Vs ACTIONNEURS

## Capteurs:

Contact	Pression	Numérique	<ul style="list-style-type: none"> <li>• Interrupteur</li> <li>• DFRobot Digital Push Button</li> </ul>
		Analogique	<ul style="list-style-type: none"> <li>• Capteur de force</li> </ul>
	Rotation	Analogue	<ul style="list-style-type: none"> <li>• Potentiomètre</li> </ul>
		TTL	<ul style="list-style-type: none"> <li>• Encodage de rotation</li> </ul>
Espace	Mouvement	Numérique	<ul style="list-style-type: none"> <li>• Capteur PIR de mouvement</li> </ul>
	Présence	Numérique	<ul style="list-style-type: none"> <li>• Tapis sensitif</li> <li>• TimeTech Laser Sensor</li> </ul>
	Proximité	Numérique	<ul style="list-style-type: none"> <li>• DFRobot Capacitive Touch Sensor</li> </ul>
		Analogique	<ul style="list-style-type: none"> <li>• IR compound eye</li> </ul>
		Intervalle	<ul style="list-style-type: none"> <li>• Capacitance</li> </ul>
		I2C	<ul style="list-style-type: none"> <li>• Capteur de chaleur 8 pixels</li> </ul>
	Distance	Analogique	<ul style="list-style-type: none"> <li>• Capteur de distance Sharp GP2D12</li> <li>• MaxSonar EZ1</li> </ul>
		Intervalle ou série	<ul style="list-style-type: none"> <li>• Capteur de Distance Ultrasonique SeeedStudio</li> </ul>
Surface	Lignes	Numérique	<ul style="list-style-type: none"> <li>• DFRobot Line Tracking Sensor</li> </ul>
		Analogique	<ul style="list-style-type: none"> <li>• DFRobot Grayscale Sensor</li> </ul>
	Couleur	TTL	<ul style="list-style-type: none"> <li>• TCS230-DB Color Sensor</li> </ul>
Lumière	Visible	Analogique	<ul style="list-style-type: none"> <li>• Photo-résistance</li> <li>• DFRobot Ambient Light Sensor</li> </ul>
	Invisible	Numérique	<ul style="list-style-type: none"> <li>• Capteur d'infrarouge</li> <li>• Télécommande</li> </ul>
Son	Amplitude	Analogique	<ul style="list-style-type: none"> <li>• Inex Sound Detector</li> </ul>
	Vibration	Analogique	<ul style="list-style-type: none"> <li>• Piezo</li> </ul>
Déplacement	Localisation	Série	<ul style="list-style-type: none"> <li>• Boussole numérique HM55B</li> <li>• Parallax GPS Module</li> </ul>
	Accélération	Série	<ul style="list-style-type: none"> <li>• NunChucky</li> <li>• DE-ACCM5G</li> </ul>
Identification	RFID	Série	<ul style="list-style-type: none"> <li>• Seeedstudio RFID</li> </ul>
Biométrie	Empreintes		<ul style="list-style-type: none"> <li>• ARA-ME-01</li> </ul>
	Alcool		<ul style="list-style-type: none"> <li>• Alcohol Gas Sensor MQ-3</li> </ul>
	Rythme cardiaque		<ul style="list-style-type: none"> <li>• Polar Heart Rate Module - RMCM01</li> </ul>
Environnement	Température	Analogique	<ul style="list-style-type: none"> <li>• Capteur de température TMP36</li> <li>• Capteur de Température Ambiente Robotics Connection</li> </ul>
	Humidité		<ul style="list-style-type: none"> <li>• Sensirion Temp/Humidity</li> </ul>
	Vent		<ul style="list-style-type: none"> <li>• VORTEX Wind Speed Sensor</li> </ul>

## Actionneurs:

Lumière	Simple	<ul style="list-style-type: none"> <li>• DEL</li> <li>• DFRobot White LED</li> <li>• BlinkM MaxM</li> <li>• Relais</li> <li>• DFRobot Single Relay</li> </ul>
	Matrice	<ul style="list-style-type: none"> <li>• Démultiplexeur</li> <li>• MAX7219</li> <li>• MondoMatrix LEDMatrix</li> <li>• Rainbowduino</li> </ul>
	PWM multiples	<ul style="list-style-type: none"> <li>• TLC5940</li> </ul>
Moteurs	Courant continu (CC)	<ul style="list-style-type: none"> <li>• Circuit de base pour un moteur CC</li> <li>• Adafruit Motor Shield</li> <li>• Relais</li> <li>• DFRobot Single Relay</li> <li>• Seeedstudio Relay Shield</li> </ul>
	Servomoteur	<ul style="list-style-type: none"> <li>• Circuit de base pour un servomoteur</li> <li>• DFRobot IO Expansion Shield</li> </ul>
	Moteur à pas	<ul style="list-style-type: none"> <li>• Circuit de base pour un moteur à pas</li> <li>• Adafruit Motor Shield</li> </ul>
	120V AC	<ul style="list-style-type: none"> <li>• Relais</li> <li>• DFRobot Single Relay</li> <li>• Seeedstudio Relay Shield</li> <li>• DMX</li> </ul>
	Solénoïde	<ul style="list-style-type: none"> <li>• Relais</li> </ul>
	Audio	<ul style="list-style-type: none"> <li>• Resistor ladder audio DAC</li> <li>• Arduino Realtime Audio Effects</li> </ul>

<http://wiki.t-o-f.info/index.php?n=Arduino.Capteur>

=> EXERCONS-NOUS SUR LES COMPOSANTS DU KIT ARDUINO !!!

## III – 1 ENTREES/SORTIES DIGITALES

# ENTREE/SORTIE NUMERIQUE

La carte ARDUINO possède 14 entrées/sorties numériques D<sub>0</sub> à D<sub>13</sub>.

Dans « void setup », il faut déclarer une broche comme une entrée ou comme une sortie par une des deux instructions suivantes:

```
pinMode (nom_de_broche, INPUT); //broche en entrée  
pinMode (nom_de_broche, OUTPUT); //broche en sortie
```

En sortie, on envoie soit 5V sur la broche, soit 0V.

Cela correspond à un « 1 » ou à un « 0 », à un niveau « haut » ou à un niveau « bas ».

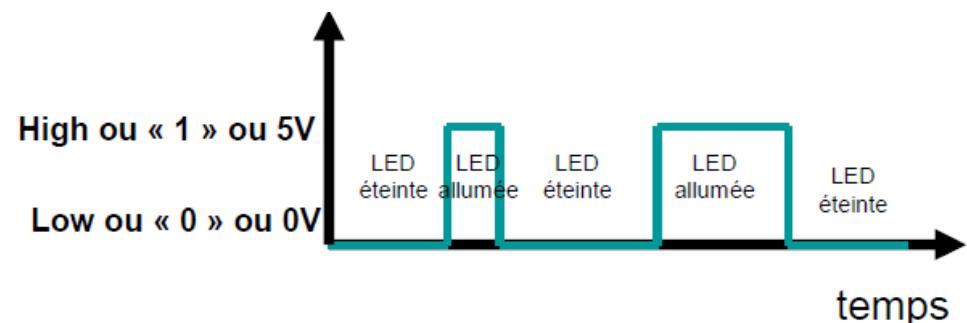
Dans le programme cela correspond aux instructions suivantes:

```
digitalWrite (nom_de_broche, HIGH); //envoie 5V sur la broche soit « 1 »  
digitalWrite (nom_de_broche, LOW); //envoie 0V sur la broche soit « 0 »
```

En entrée, la carte peut lire soit un niveau haut (« 1 » ou HIGH) soit un niveau bas (« 0 » ou LOW)

Dans le programme cela correspond aux instructions suivantes:

```
digitalRead (nom_de_broche);
```



**Signal numérique** : signal qui ne prend que deux états distinct comme 0V et 5V soit « 0 » et « 1 ».

Pour mémoire, on trouve ici la tension des niveaux logiques d'entrée pour la carte Arduino (ATmega 328) :

## 28.2 DC Characteristics

$T_A = -40^{\circ}\text{C}$  to  $85^{\circ}\text{C}$ ,  $V_{CC} = 1.8\text{V}$  to  $5.5\text{V}$  (unless otherwise noted)

Symbol	Parameter	Condition	Min.	Typ.	Max.	Units
$V_{IL}$	Input Low Voltage, except XTAL1 and RESET pin	$V_{CC} = 1.8\text{V} - 2.4\text{V}$ $V_{CC} = 2.4\text{V} - 5.5\text{V}$	-0.5		$0.2V_{CC}^{(1)}$ $0.3V_{CC}^{(1)}$	V
$V_{IH}$	Input High Voltage, except XTAL1 and RESET pins	$V_{CC} = 1.8\text{V} - 2.4\text{V}$ $V_{CC} = 2.4\text{V} - 5.5\text{V}$	$0.7V_{CC}^{(2)}$ $0.6V_{CC}^{(2)}$		$V_{CC} + 0.5$ $V_{CC} + 0.5$	V

Ce qui donne en  $V_{CC}=5\text{V}$  :

- Niveau Logique BAS (LOW) en entrée :  $V_{IL} = 0.3 \times V_{CC} = 0.3 \times 5\text{V} = 1.5\text{ Volts}$ .

Une tension comprise entre -0.5 et inférieure à 1.5V est considérée comme un niveau BAS.

- Niveau Logique HAUT (HIGH) en entrée :  $V_{IH} = 0.6 \times V_{CC} = 0.6 \times 5\text{V} = 3.0\text{ Volts}$ .

Une tension supérieure à 3V et inférieure à 5.5V est considérée comme un niveau HAUT.

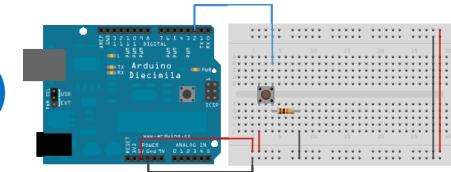
- La zone intermédiaire 1,5V - 3V est indéterminée et doit être évitée... Les niveaux HAUT / BAS acceptés ont des valeurs assez tolérantes.

3 définitions possibles avec l'instruction pinMode () :

- INPUT,
- INPUT\_PULLUP et
- OUTPUT

#### Pins configurées comme entrée:

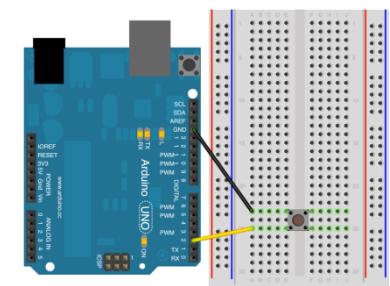
- **INPUT:** Les broches Arduino (Atmega) sont configurées en entrée avec pinMode () dans un état de haute impédance (résistance série de  $100\text{ M}\Omega$ )  
- Exemple: pinMode (inPin, INPUT);



Cependant, vous voulez sûrement que la broche soit référencée à la terre, donc 2 solutions:

- Soit mettre une résistance de pull-down (une résistance allant à la terre) dans le circuit extérieur
- Soit configurer la pin entrée en tant que INPUT\_PULLUP, qui possède une résistance pull-up

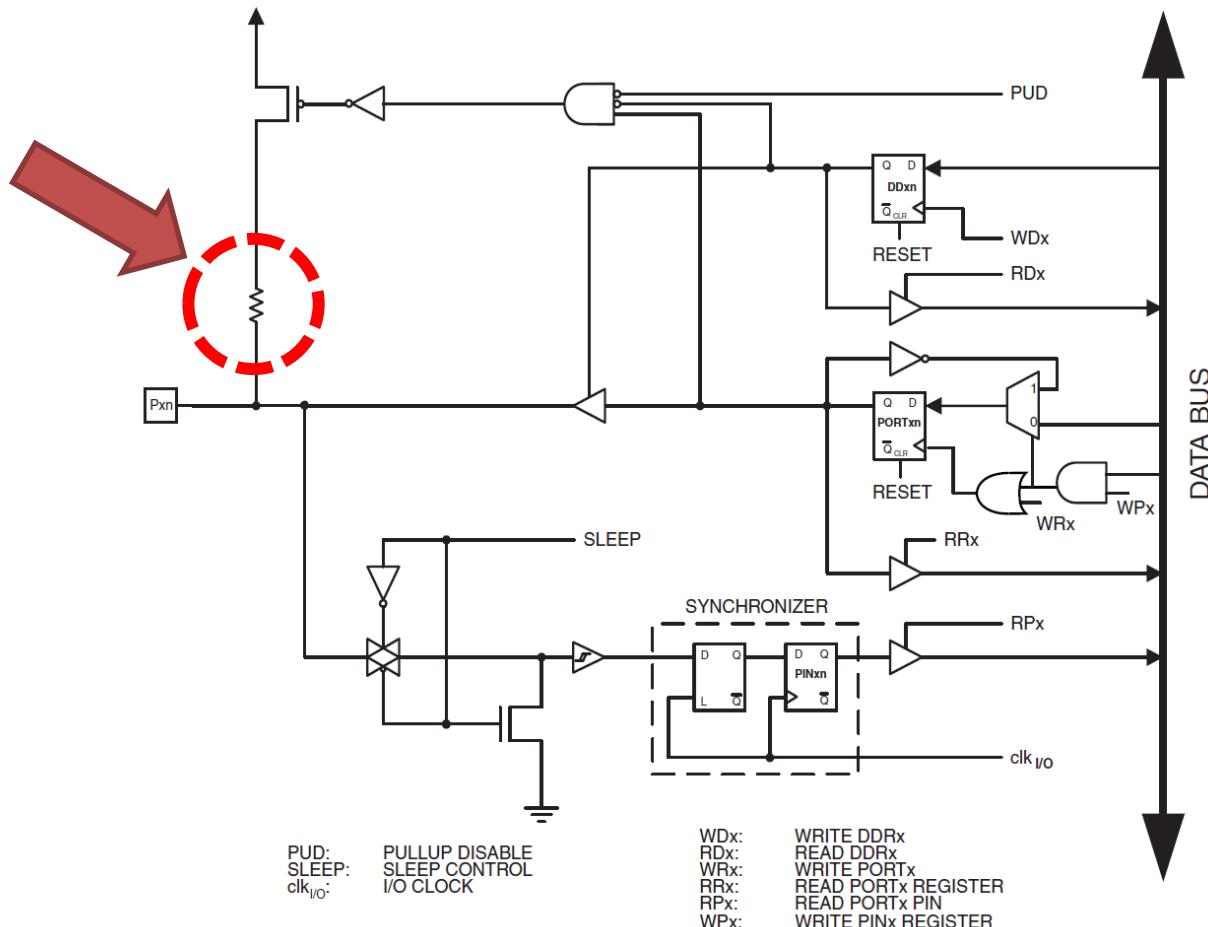
- **INPUT\_PULLUP:** la puce Atmega sur l'Arduino possède une résistance de pull-up interne. Attention, puisque pull-up => cela inverse le comportement de la pin, puisque HIGH signifie que le capteur est « éteint », et LOW signifie que le capteur est activé.  
- Exemple: pinMode (2, INPUT\_PULLUP);



#### Pins configurées en sorties:

- **OUTPUT:** Les pins configurées comme sortie avec pinMode () sont censées être dans un état de faible impédance. Cela signifie qu'elles peuvent fournir une quantité importante de courant à d'autres circuits.  
Exemple: pinMode (2, OUTPUT);

## POUR ALLER PLUS LOIN

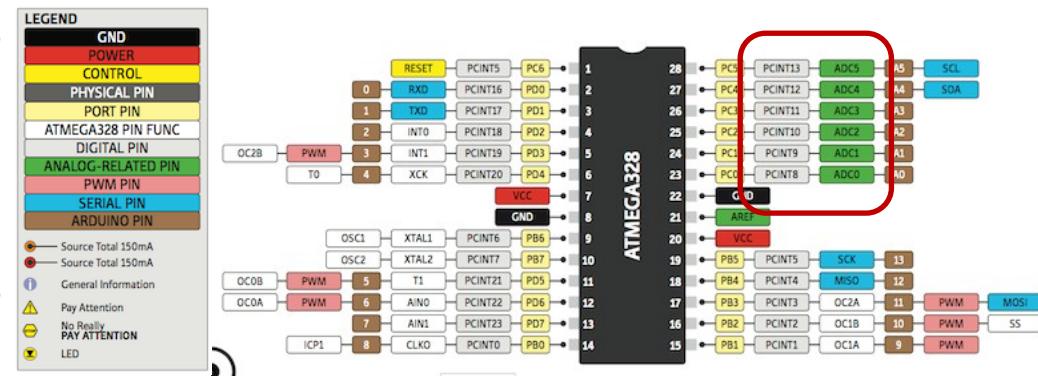
Figure 14-2. General Digital I/O<sup>(1)</sup>

L'ATmega de l'Arduino a 6 canaux analogique-numériques (A/N) (8 en version CMS), pour 10 bits de résolution

Bien que la fonction principale des broches analogiques soit de lire les capteurs en ENTREES analogiques

Elles ont également toutes les fonctionnalités d'usage général des broches d'entrée/sortie (GPIO) => les mêmes que les broches DIGITALES (0 à 13).

=> si un utilisateur a besoin de plus de broches d'entrée/sortie DIGITALES: il peut utiliser les broches analogiques comme GPIO.



## COMMENT FAUT-IL FAIRE ?

Il faut alors utiliser l'alias A0 (pour l'entrée analogique 0), A1, etc...

*Par exemple, pour utiliser la pin A0 en SORTIE DIGITALE, le code ressemble à ceci:*

**pinMode (A0, OUTPUT);**

**digitalWrite (A0, HIGH); //pullup mise sur la broche analogique 0**

La résistance de Pull-up est alors activée

### Attention:

1 - les autres broches analogiques ont aussi une résistance de pull-up, qui fonctionne comme celle des broches numériques. Elles sont activées par la commande:

**digitalWrite (A0, HIGH); //pullup mise sur la broche analogique 0**

**même si la broche est déclarée en entrée.**

2 - l'activation d'un pullup aura une incidence sur les valeurs déclarées par la fonction **analogRead()**, utilisée sur le PORT C des pins analogiques (**ATTENTION**).

Les DEL (diodes électroluminescentes) sont très utilisées de nos jours.

Nous allons commencer avec quelque chose de très simple: allumer et éteindre, à plusieurs reprises => clignotement C'est le « HELLO WORLD » des µ-contrôleurs !

```
/* Blink
 *Turns on an LED on for one second, then off for one second, repeatedly.
 *The circuit: LED connected from digital pin 13 to ground.
 */
```

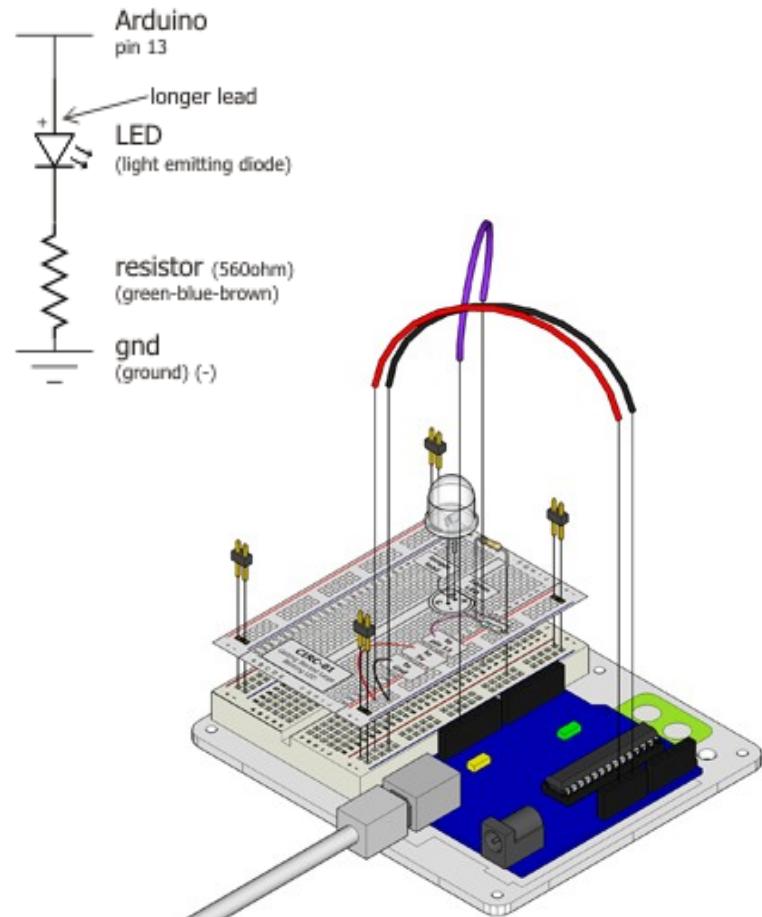
```
int ledPin = 13; // LED connected to digital pin 13

// The setup() method runs once, when the sketch starts

void setup() {
  pinMode(ledPin, OUTPUT); // initialize the digital pin as an output:
}

// the loop() method runs over and over again, as long as the Arduino has
power

void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  delay(1000); // wait for a second
  digitalWrite(ledPin, LOW); // set the LED off
  delay(1000); // wait for a second
}
```



<http://www.oomlout.com/a/products/ardx/circ-01>

← RETOUR

Une fois votre programme téléchargé dans Arduino, le programme fonctionne en toute autonomie. Tout le monde est content s'il fonctionne comme attendu... **par contre, si ce n'est pas le cas, il faut commencer à comprendre ce qui se passe.**

Il faut faire preuve d'imagination et élaborer des hypothèses... et il faut bien avouer que c'est une tâche fastidieuse car **le board ne possède ni debugger, ni affichage de message de debugging!**

**Pourtant il y a le port série...** il est donc possible d'envoyer des messages (*message de debugging*) depuis Arduino et de les lire sur le PC.

## Port série Arduino et Port USB:

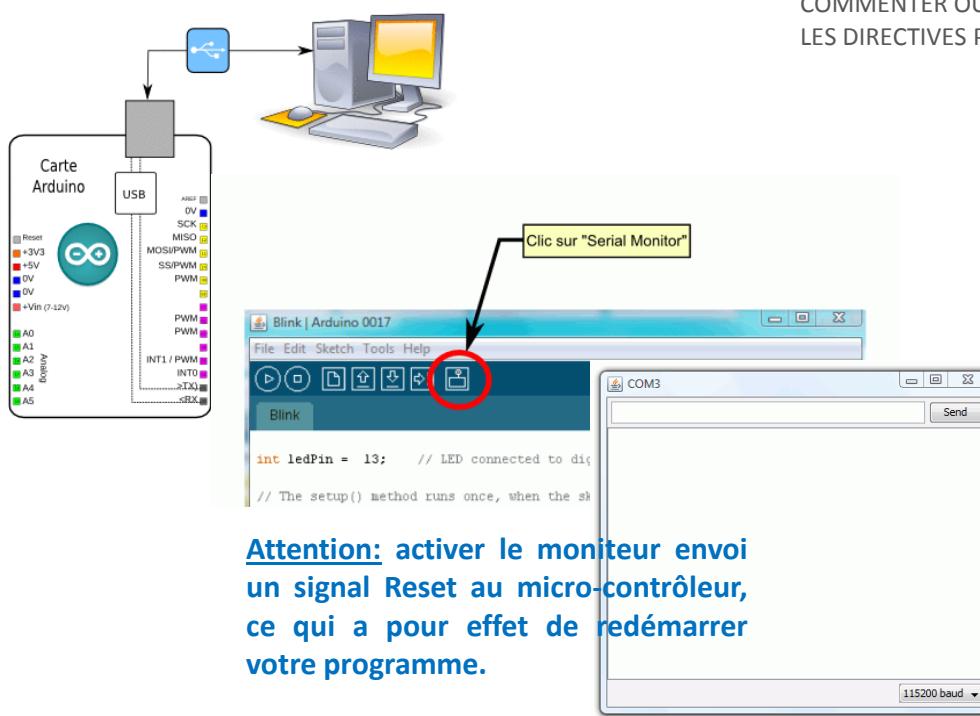
Il est possible de communiquer via le port série d'Arduino (c'est ce que le micro-contrôleur croit) **mais en fait, tous les messages sont renvoyés via le périphérique USB (cf. FTDI) vers le PC (et son port USB).**  
Ne reste plus qu'à écouter le port USB.

## Attention: réservation des pins 0 et 1 pour la communication série:

Lors d'une connexion série, les **Pin0 et 1 sont automatiquement re-configurées en RX (pin0) & TX (pin1).**  
=> il devient impossible d'utiliser les pins 0 & 1 à d'autres fins.

**Note: c'est d'ailleurs cette même connexion USB sert également à alimenter la board.**

# AFFICHE DU TEXTE SUR LE PORT SERIE => DEBUGGAGE



**Attention:** activer le moniteur envoi un signal Reset au micro-contrôleur, ce qui a pour effet de redémarrer votre programme.

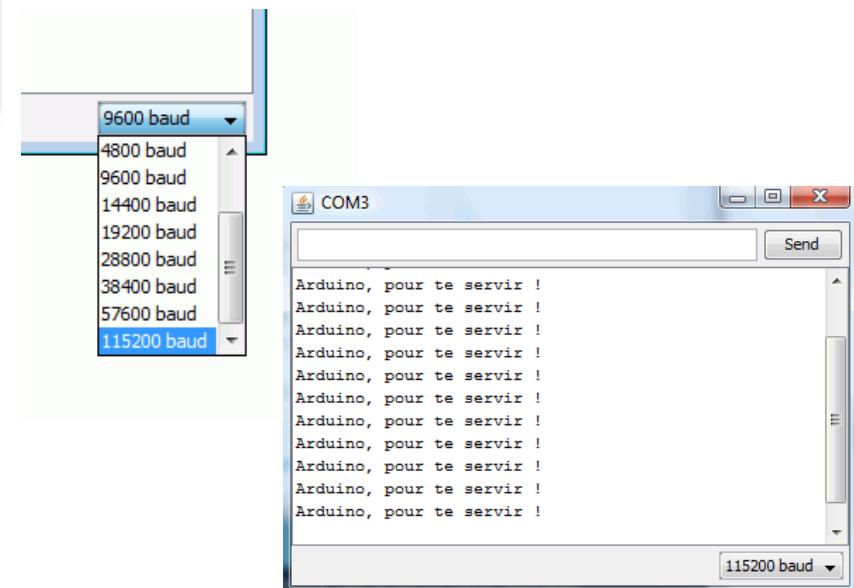
COMMENTER OU DECOMMENTER  
LES DIRECTIVES PRE-PROCESSEUR →

```
#define DEBUG

#ifndef DEBUG
unsigned int mavar =255;
#endif

void setup() {
#ifndef DEBUG
Serial.begin(9600); //Initialisation port Série
#endif
}

void loop() {
#ifndef DEBUG
int droite=digitalRead(CNY_D); //TEST CAPTEUR CNY70
Serial.print(" G:"); Serial.print(gauche);
Serial.print(", BP:");
Serial.println(digitalRead(BP)); //TEST BOUTON POUSSOIR
#endif
} // fin de loop
```



```
/* Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 * The circuit: LED connected from digital pin 13 to ground.
 * Note: On most Arduino boards, there is already an LED on the board
 * connected to pin 13, so you don't need any extra components for this example.
```

#### + Affiche l'état de la LED dans le Sérial moniteur

\*Created 1 June 2005, By David Cuartielles  
<http://arduino.cc/en/Tutorial/Blink>

```
/*
int ledPin = 13; // LED connected to digital pin 13
```

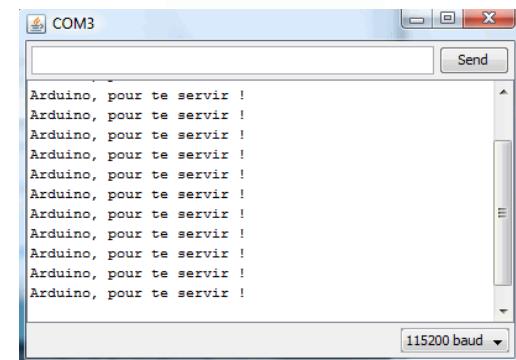
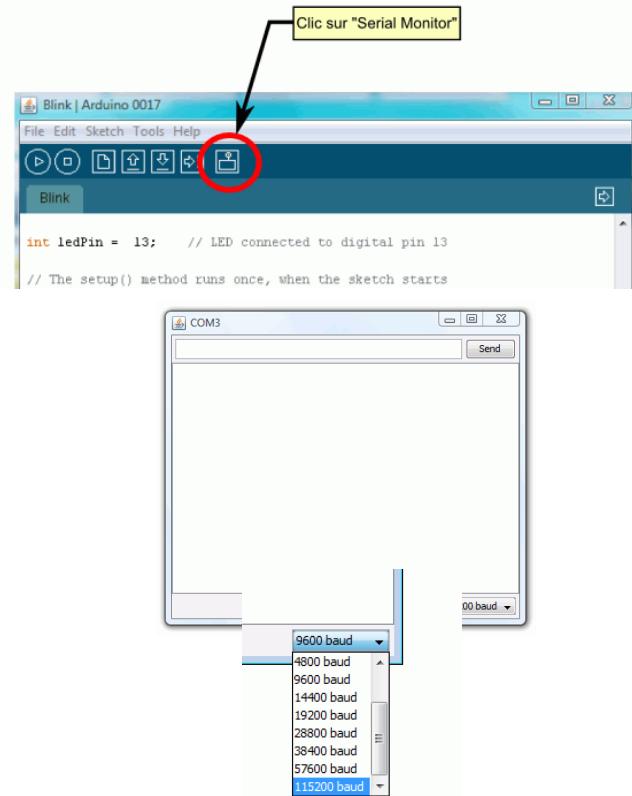
// The setup() method runs once, when the sketch starts

```
void setup() {
  // opens serial port, sets data rate to 9600 bps
  Serial.begin(9600);
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
}
```

// the loop() method runs over and over again, as long as the Arduino has power

```
void loop()
{
  digitalWrite(ledPin, HIGH); // set the LED on
  Serial.print("LED Status: ON\n");
  delay(1000); // wait for a second

  digitalWrite(ledPin, LOW); // set the LED off
  Serial.print("LED Status: OFF\n");
  delay(1000); // wait for a second
}
```



Obtenir le nombre d'octets (caractères) disponibles pour la lecture à partir du port série.  
Il s'agit de données qui sont déjà arrivées et stockées dans le buffer de réception série (qui détient 128 octets).

```
int incomingByte = 0; // for incoming serial data
```

```
void setup() {
Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}
```

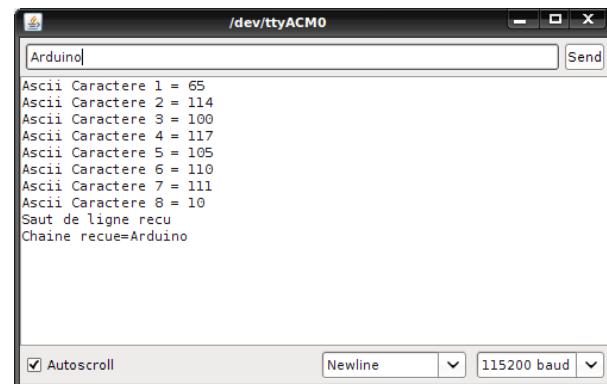
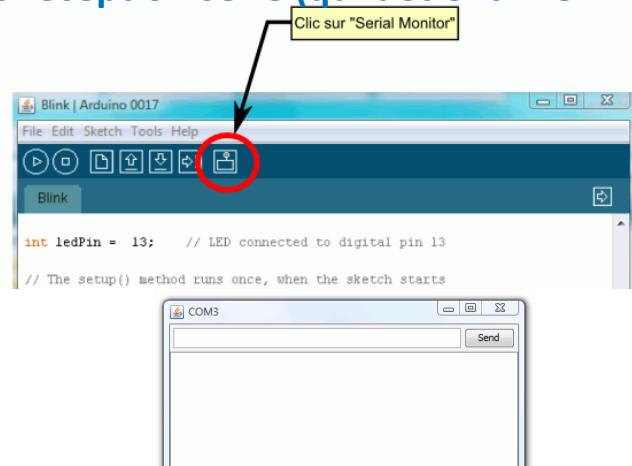
```
void loop() {

// send data only when you receive data:
if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();

    // say what you got:
    Serial.print("I received: ");
    Serial.println(incomingByte, DEC);
}

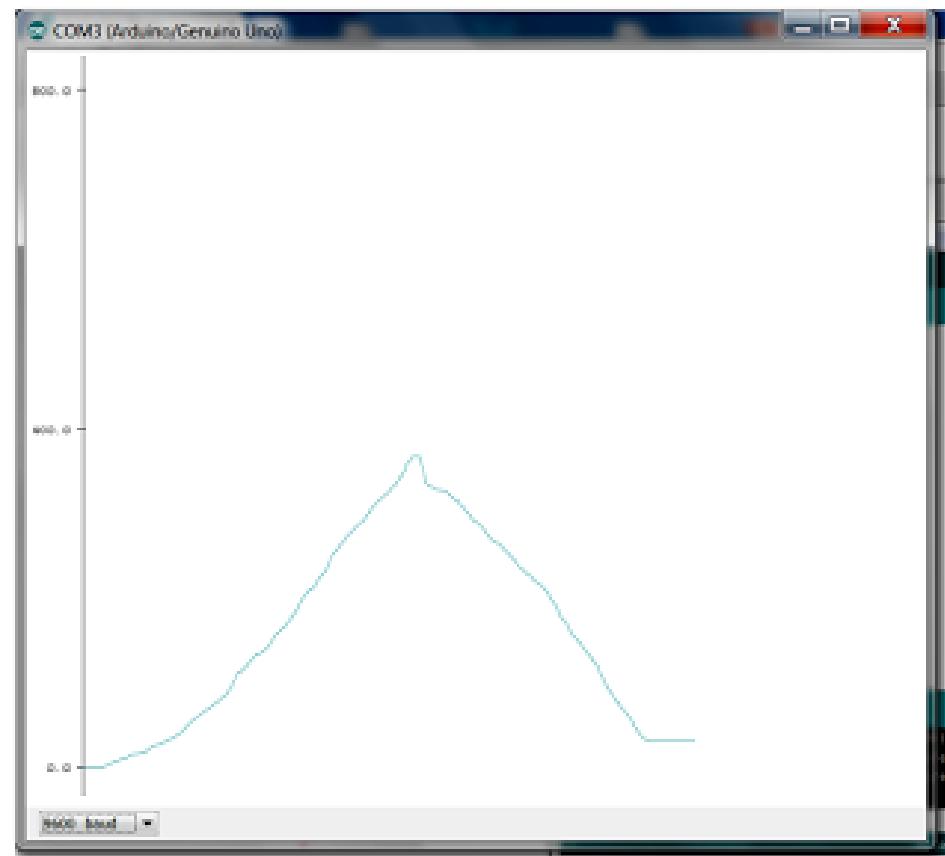
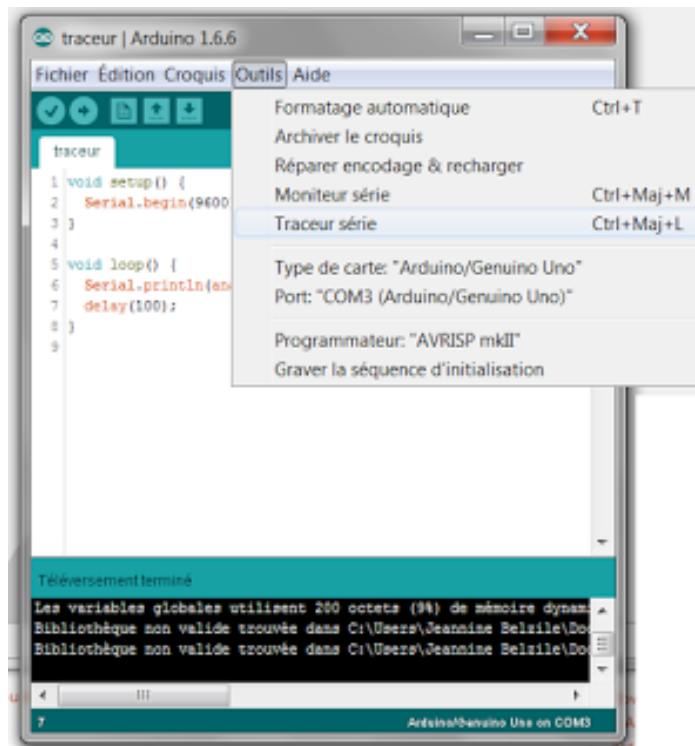
}
```

<http://www.arduino.cc/en/Serial/Available>





```
1 void setup() {  
2     Serial.begin(9600);  
3 }  
4  
5 void loop() {  
6     Serial.println(analogRead(A0));  
7     delay(100);  
8 }
```



Le traceur série affiche un nombre fixe de données. Il est important de bien régler votre `delay()` dans la boucle.

**Exercice:**

Reprenez le montage du premier programme et provoquez l'allumage ou l'extinction de la LED par l'envoi du caractère y et n respectivement.

```
/* Blink
 * Turns on an LED on for one second, then off for one second, repeatedly.
 * The circuit: LED connected from digital pin 13 to ground.
 * Created 1 June 2005, By David Cuartielles
 * http://arduino.cc/en/Tutorial/Blink
 */
```

```
int ledPin = 13; // LED connected to digital pin 13
int incomingByte = 0; //serial char
```

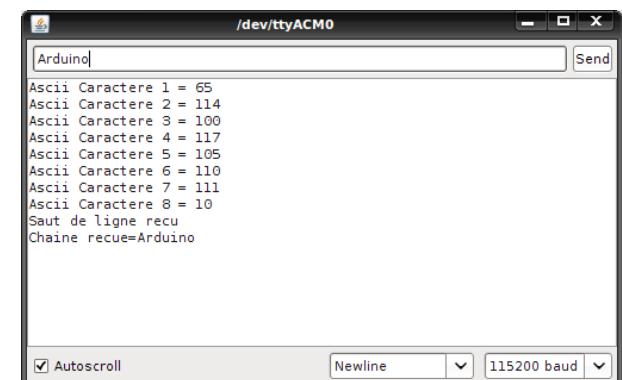
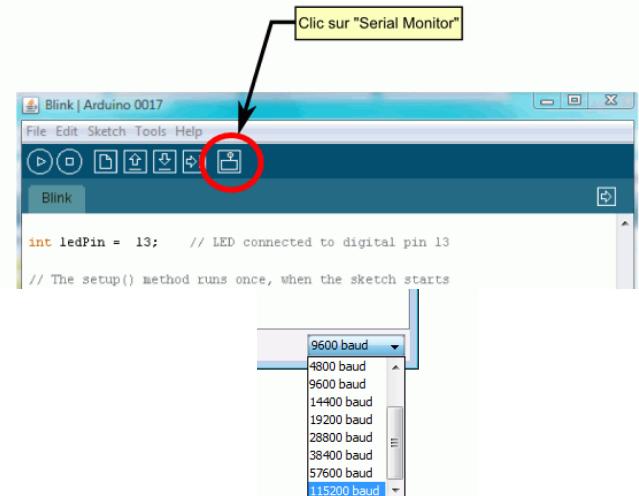
// The setup() method runs once, when the sketch starts

```
void setup() {
  // opens serial port, sets data rate to 9600 bps
  Serial.begin(9600);
  // initialize the digital pin as an output:
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);
}
```

// the loop() method runs over and over again, as long as the Arduino has power

```
void loop()
{
  if (Serial.available() > 0) {
    // read the incoming byte:
    incomingByte = Serial.read();
```

```
    if (incomingByte == 'y') {
      digitalWrite(ledPin, HIGH); // set the LED ON
    }
    else if (incomingByte == 'n') {
      digitalWrite(ledPin, LOW); // set the LED off
    }
  }
  delay(250); // wait for a second
}
```



Comment faire pour débugger ? => la librairie Série « LOGICIELLE »: SoftwareSerial

*Par exemple: module GPS*

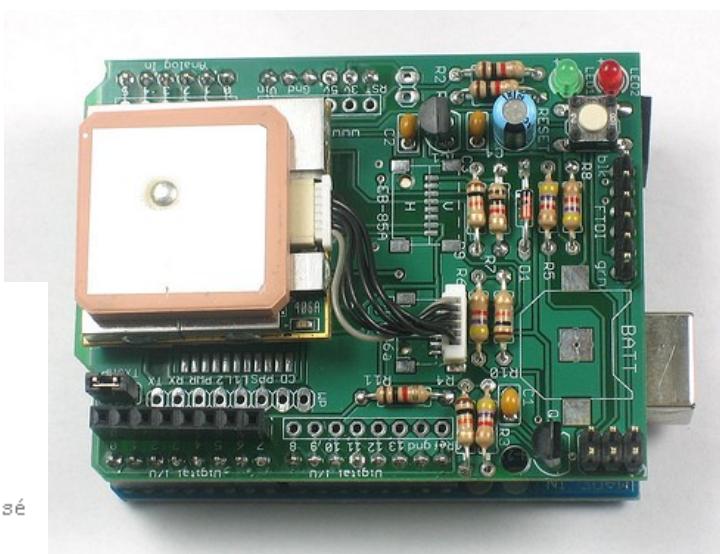
- Pin 0 & Pin1 occupée pour la communication Série  
=> on définit la Pin 2 & Pin3 (par exemple)  
pour afficher les données du module GPS sur le Serial Monitor

```
/* Ce programme fait afficher sur le moniteur série de
l'environnement de développement toutes les trames émises
par le récepteur GPS
*/
#include <SoftwareSerial.h>

SoftwareSerial PortSerie(2,3); // Tx du GPS sur 2 de l'Arduino, Rx sur 3 mais non utilisé

void setup()
{
    Serial.begin(115200); // Initialisation du port série matériel
    PortSerie.begin(4800); // Initialisation du port série logiciel
    Serial.println("Affichage des trames du GPS");
    Serial.println();
}

void loop()
{
    while (PortSerie.available()) // Tant que des données GPS sont disponibles
    {
        char c = PortSerie.read(); // Lecture de ces données sur le port série logiciel
        Serial.write(c); // Affichage sur le port série matériel
    }
}
```



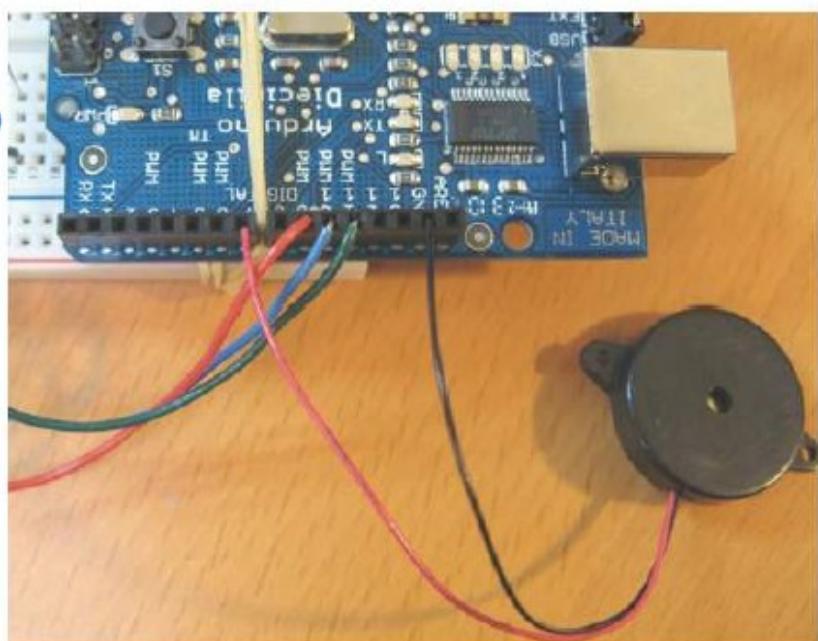
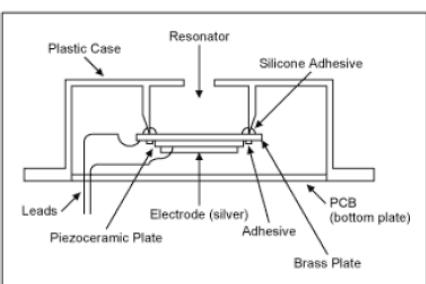
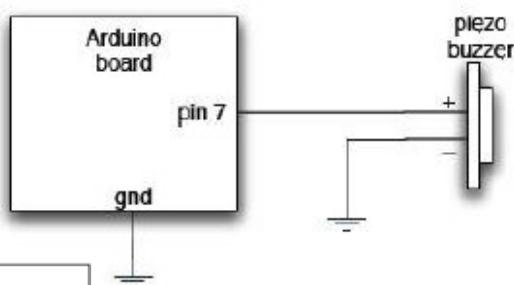
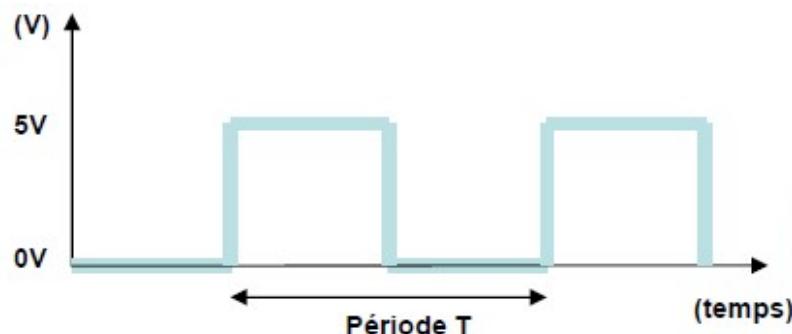
*Bien sûr, l'affichage sur le Serial monitor doit aller moins vite que l'acquisition de données par le module!!!*

# CABLER UNE BUZZER (PIEZO SPEAKER)

**Le buzzer se câble sur une sortie numérique.**

**On lui envoie alors un signal périodique dont on fait varier la fréquence en fonction de la note que l'on désire jouer:**

**Exemple: le LA est un signal de fréquence  $f$  de 440Hz**



	0	1	2	3	4	5	6	7	8	9
do	32.703	65.406	130.81	261.63	523.25	1046.5	2093.	4186.	8372.	16744.
dot	34.648	69.296	138.59	277.18	554.37	1108.7	2217.5	4434.9	8869.8	17740.
ré	36.708	73.416	146.83	293.66	587.33	1174.7	2349.3	4698.6	9397.3	18795.
ré#	38.891	77.782	155.56	311.13	622.25	1244.5	2489.	4978.	9956.1	19912.
mi	41.203	82.407	164.81	329.63	659.26	1318.5	2637.	5274.	10548.	21096.
fa	43.654	87.307	174.61	349.23	698.46	1396.9	2793.8	5587.7	11175.	22351.
fa#	46.249	92.499	185.	369.99	739.99	1480.	2960.	5919.9	11840.	23680.
sol	48.999	97.999	196.	392.	783.99	1568.	3136.	6271.9	12544.	25088.
sol#	51.913	103.83	207.65	415.3	830.61	1661.2	3322.4	6644.9	13290.	26580.
la	55	110	220	440	880	1760	3520	7040	14080	28160
lat#	58.27	116.54	233.08	466.16	932.33	1864.7	3729.3	7458.6	14917.	29834.
si	61.735	123.47	246.94	493.88	987.77	1975.5	3951.1	7902.1	15804.	31609.

$$f = 1/T ; T=1/f ; f : \text{fréquence}; T : \text{période}.$$

# CABLER UNE BUZZER (PIEZO SPEAKER) - PROGRAMME

```
/* Melody (copyleft) 2005 D. Cuartielles for K3
```

\* This example uses a piezo speaker to play melodies. It sends a square wave of the appropriate frequency to the piezo, generating

\* the corresponding tone. The calculation of the tones is made following the mathematical operation: timeHigh = period / 2 = 1 / (2 \* toneFrequency)

\* where the different tones are described as in the table:

\* note frequency period timeHigh

* c	261 Hz	3830 1915
-----	--------	-----------

* d	294 Hz	3400 1700
-----	--------	-----------

* e	329 Hz	3038 1519
-----	--------	-----------

* f	349 Hz	2864 1432
-----	--------	-----------

* g	392 Hz	2550 1275
-----	--------	-----------

* a	440 Hz	2272 1136
-----	--------	-----------

* b	493 Hz	2028 1014
-----	--------	-----------

* C	523 Hz	1912 956
-----	--------	----------

\* http://www.arduino.cc/en/Tutorial/Melody

\*/

```
int speakerPin = 9;
```

```
int length = 15; // the number of notes
```

```
char notes[] = "ccggaagfeeddc"; // a space represents a rest
```

```
int beats[] = { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 2, 4 };
```

```
int tempo = 300;
```

```
void setup() {
```

```
  pinMode(speakerPin, OUTPUT);
```

```
}
```

```
void loop() {
```

```
  for (int i = 0; i < length; i++) {
```

```
    if (notes[i] == ' ') { delay(beats[i] * tempo); // rest
    } else { playNote(notes[i], beats[i] * tempo);
    }
```

```
    delay(tempo / 2); // pause between notes
  }
}
```

```
void playTone(int tone, int duration) {
```

```
  for (long i = 0; i < duration * 1000L; i += tone * 2) {
    digitalWrite(speakerPin, HIGH); delayMicroseconds(tone);
    digitalWrite(speakerPin, LOW); delayMicroseconds(tone);
  }
}
```

```
void playNote(char note, int duration) {
```

char names[] = { 'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C' };
--

int tones[] = { 1915, 1700, 1519, 1432, 1275, 1136, 1014, 956 };
--

// play the tone corresponding to the note name

```
  for (int i = 0; i < 8; i++) {
```

```
    if (names[i] == note) {
```

```
      playTone(tones[i], duration);
    }
  }
}
```

The diagram shows a simple circuit. An Arduino pin labeled "Arduino pin 9" is connected to one terminal of a piezo element. The other terminal of the piezo element is connected to ground, labeled "gnd (ground) (-)".

<http://www.oomlout.com/a/products/ardx/circ-06>

Musical notation for the first section of the melody, starting with a treble clef and a common time signature.

Musical notation for the second section of the melody, continuing from the previous section.

A 3D rendering of an Arduino Uno microcontroller board. A breadboard is mounted on top of it. A piezo speaker is connected to the breadboard. Various wires and components are visible, illustrating the physical setup of the project.

Musical notation for the third section of the melody, continuing from the previous sections.

Musical notation for the fourth section of the melody, continuing from the previous sections.

 RETOUR

J. GRISOLIA (DGP) : MICROCONTROLEURS & OPEN SOURCE HARDWARE

85

### III – 2 ENTREES ANALOGIQUES

III – 2 ENTREES ANALOGIQUES

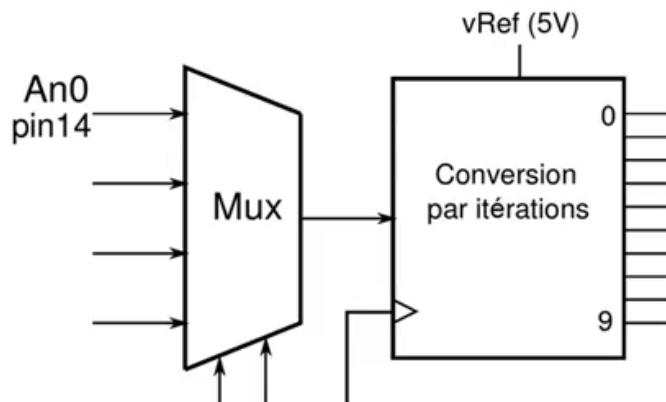
# LES ENTREES ANALOGIQUES

Contrairement au signal numérique qui ne peut prendre que deux états différents, un signal analogique peut prendre une infinité de valeurs, comme une tension que l'on fait varier progressivement de 0V à 5V.

La carte Arduino fonctionne en numérique, le microcontrôleur ne comprend que les « 0 » et les « 1 ».

Les entrées analogiques A<sub>0</sub> à A<sub>5</sub> sont donc dotées de convertisseurs analogique/numérique qui convertissent une tension en une suite de « 0 » et de « 1 » que la carte fait correspondre à un nombre de 0 à 1023 (codage sur 10 bits)

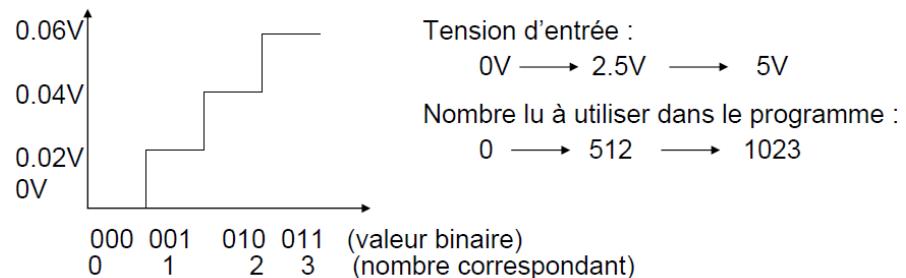
On peut ainsi récupérer les informations d'un capteur.



Valeur\_lue = analogRead(AN0);

```

int valeurLue ; byte valeurUtile;
valeurLue = analogRead (AN0) ; //10 bits
map (valeurLue, minA,maxA,minV,maxV) ;
  
```

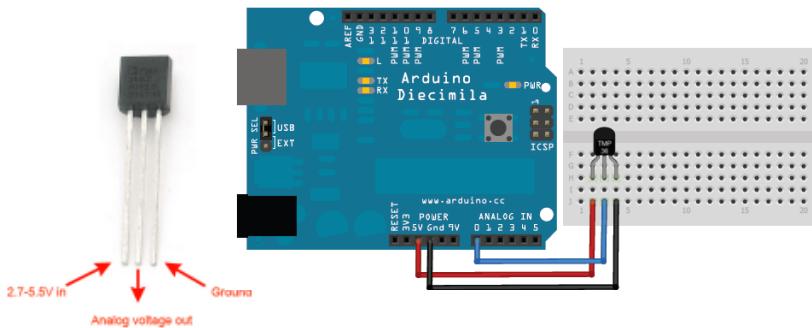


La précision dépend du câblage, difficile de garantir les 2 derniers bits!

La conversion dure 110 microsecondes (AT328)  
 Un flag dit quand elle est terminée

# CAPTEUR DE TEMPERATURE

Le TMP36 est un capteur qui se présente sous la forme d'un petit transistor.  
Une fois alimenté (entre 3 et 5V), il sort une tension analogique directement proportionnelle à la T.



## SPECIFICATIONS

$V_s = 2.7 \text{ V to } 5.5 \text{ V}$ ,  $-40^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$ , unless otherwise noted.

Parameter <sup>1</sup>	Symbol	Test Conditions/Comments	Min	Typ	Max	Unit
ACCURACY						
TMP35/TMP36/TMP37 (F Grade)		$T_A = 25^\circ\text{C}$		$\pm 1$	$\pm 2$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (G Grade)		$T_A = 25^\circ\text{C}$		$\pm 1$	$\pm 3$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (F Grade)		Over rated temperature		$\pm 2$	$\pm 3$	$^\circ\text{C}$
TMP35/TMP36/TMP37 (G Grade)		Over rated temperature		$\pm 2$	$\pm 4$	$^\circ\text{C}$
Scale Factor, TMP35		$10^\circ\text{C} \leq T_A \leq 125^\circ\text{C}$		10		$\text{mV}/^\circ\text{C}$
Scale Factor, TMP36		$-40^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$		10		$\text{mV}/^\circ\text{C}$
Scale Factor, TMP37		$5^\circ\text{C} \leq T_A \leq 85^\circ\text{C}$		20		$\text{mV}/^\circ\text{C}$
Load Regulation		$5^\circ\text{C} \leq T_A \leq 100^\circ\text{C}$		20		$\text{mV}/^\circ\text{C}$
		$3.0 \text{ V} \leq V_s \leq 5.5 \text{ V}$				
		$0 \mu\text{A} \leq I_L \leq 50 \mu\text{A}$				$\text{mV}/\mu\text{A}$
		$-40^\circ\text{C} \leq T_A \leq +105^\circ\text{C}$	6	20		$\text{mV}/\mu\text{A}$
		$-105^\circ\text{C} \leq T_A \leq +125^\circ\text{C}$	25	60		$\text{mV}/\mu\text{A}$
Power Supply Rejection Ratio	PSRR	$T_A = 25^\circ\text{C}$		30	100	$\text{m}^\circ\text{C}/\text{V}$
Linearity		$3.0 \text{ V} \leq V_s \leq 5.5 \text{ V}$		50		$\text{m}^\circ\text{C}/\text{V}$
Long-Term Stability		$T_A = 150^\circ\text{C}$ for 1 kHz		0.5		$^\circ\text{C}$
				0.4		$^\circ\text{C}$
SHUTDOWN						
Logic High Input Voltage	$V_{\text{H}}$	$V_s = 2.7 \text{ V}$	1.8			$\text{V}$
Logic Low Input Voltage	$V_{\text{L}}$	$V_s = 5.5 \text{ V}$		400		$\text{mV}$
OUTPUT						
TMP35 Output Voltage		$T_A = 25^\circ\text{C}$		250		$\text{mV}$
TMP36 Output Voltage		$T_A = 25^\circ\text{C}$		750		$\text{mV}$
TMP37 Output Voltage		$T_A = 25^\circ\text{C}$		500		$\text{mV}$
Output Voltage Range			100		2000	$\text{mV}$
Output Load Current	$I_L$		0		50	$\mu\text{A}$
Short-Circuit Current	$I_{\text{SC}}$	Note 2			250	$\mu\text{A}$
Capacitive Load Driving	$C_L$	No oscillations <sup>2</sup>	1000	10000		$\mu\text{A}$
Device Turn-On Time		Output within $\pm 1^\circ\text{C}$ , $100 \text{ k}\Omega/\text{100 pF load}^2$	0.5	1		$\text{pF}$
						$\text{ms}$
POWER SUPPLY						
Supply Range	$V_s$		2.7		5.5	$\text{V}$
Supply Current (On)	$I_{\text{S}}(\text{ON})$	Unloaded			50	$\mu\text{A}$
Supply Current (Off)	$I_{\text{S}}(\text{OFF})$	Unloaded		0.01	0.5	$\mu\text{A}$

<sup>1</sup> Does not consider errors caused by self-heating.

<sup>2</sup> Guaranteed but not tested.

Les avantages principaux du TMP36 sont:

- 1 - Large plage de température (de  $-40$  à  $+150$  °C).
- 2 - La tension de sortie est totalement indépendante de l'alimentation du TMP36.

Caractéristique du TMP36:

Taille: boîtier TO-92 à 3 broches (similaire à un transistor)

Tension de sortie:  $0.1\text{V} (-40^\circ\text{C})$  to  $2.0\text{V} (150^\circ\text{C})$  mais la précision diminue après  $125^\circ\text{C}$

Facteur d'échelle:  $10\text{mV}/^\circ\text{C}$

Tension d'alimentation:  $2.7\text{V}$  a  $5.5\text{V}$

Courant de charge:  $0.05 \text{ mA}$

Convertir la tension analogique en degré:

Attention, le TMP36 permet de mesurer des températures négatives ! => le  $0^\circ\text{C}$  est donc placé à une offset de 500 millivolts.  
=> Si  $V_{\text{out}} < 500 \text{ mV}$  => température négative.

$$T (\text{ }^\circ\text{C}) = (V_{\text{out}}(\text{mV}) - 500) / \text{Facteur d'échelle} \Rightarrow$$

$$T (\text{ }^\circ\text{C}) = (V_{\text{out}}(\text{mV}) - 500) / 10$$

Exemple: si  $V_{\text{out}} = 1 \text{ Volts}$ ,  $T(\text{ }^\circ\text{C}) = (1000 - 500)/10 = 50^\circ\text{C}$

Équivalents aux TMP36:

LM35/TMP35 (mesure en °C) ou LM34/TMP34 (mesure en °F).

Attention aux formules de conversion: le LM35 n'a pas les 500mV d'offset

L'entrée analogique prend une valeur entre 0 et 1023 pour une tension variant entre 0 et 5V.

Précision de mesure:  $5/2^{10}=5/1024=0.0048V$  ( $\sim 4.8\text{ mV}$ ).

=> Précision de l'entrée analogique: environ +/- 0.5 °C (e.g. pour le TMP36:  $4.8(\text{mV})/10(\text{mV}/\text{°C})$ ).

### Comment augmenter la précision de l'entrée analogique ?

Par la fonction `analogReference(type)` qui configure la tension utilisée pour analog input:

**DEFAULT:** analog reference par défaut = **5V ou 3.3V**

**INTERNAL:** une référence interne fournie **1.1V** pour les ATmega168, ATmega328

**EXTERNAL:** la tension utilisée par la pin AREF (**0 à 5V**).

#### Référence EXTERNE:

En alimentant en **3.3V** (à partir de l'Arduino) et en utilisant cette tension comme référence (**pin AREF**) pour les lectures analogiques, on améliore la précision de lecture:

En effet, l'entrée analogique évoluera de 0 à 1023 pour une tension évoluant entre 0 et ARef (3.3V).

=> Précision de mesure:  $3.3/2^{10}=3/1024\sim 3.2\text{mV}$  => l'erreur est ici réduite à +/- 0.33°C.

#### Référence INTERNE:

En alimentant en **1.1V**, on améliore la précision de lecture:

=> En effet, l'entrée analogique évoluera de 0 à 1023 pour une tension évoluant entre 0 et ARef (1.1V).

=> Précision de mesure:  $1.1/2^{10}=1.1/1024\sim 1.1\text{mV}$  => erreur est ici réduite à +/- 0.1°C.

Dans la pratique: soit **analogReference(EXTERNAL)** ou **analogReference(INTERNAL)** dans **setup()**.

Attention, ne pas oublier de modifier la ligne de code calculant la conversion de `analogRead` en tension.

La ligne de code passe de `float tension = V*5.0;` à `float tension = V*3.3;` ou `float tension = V*1.1`

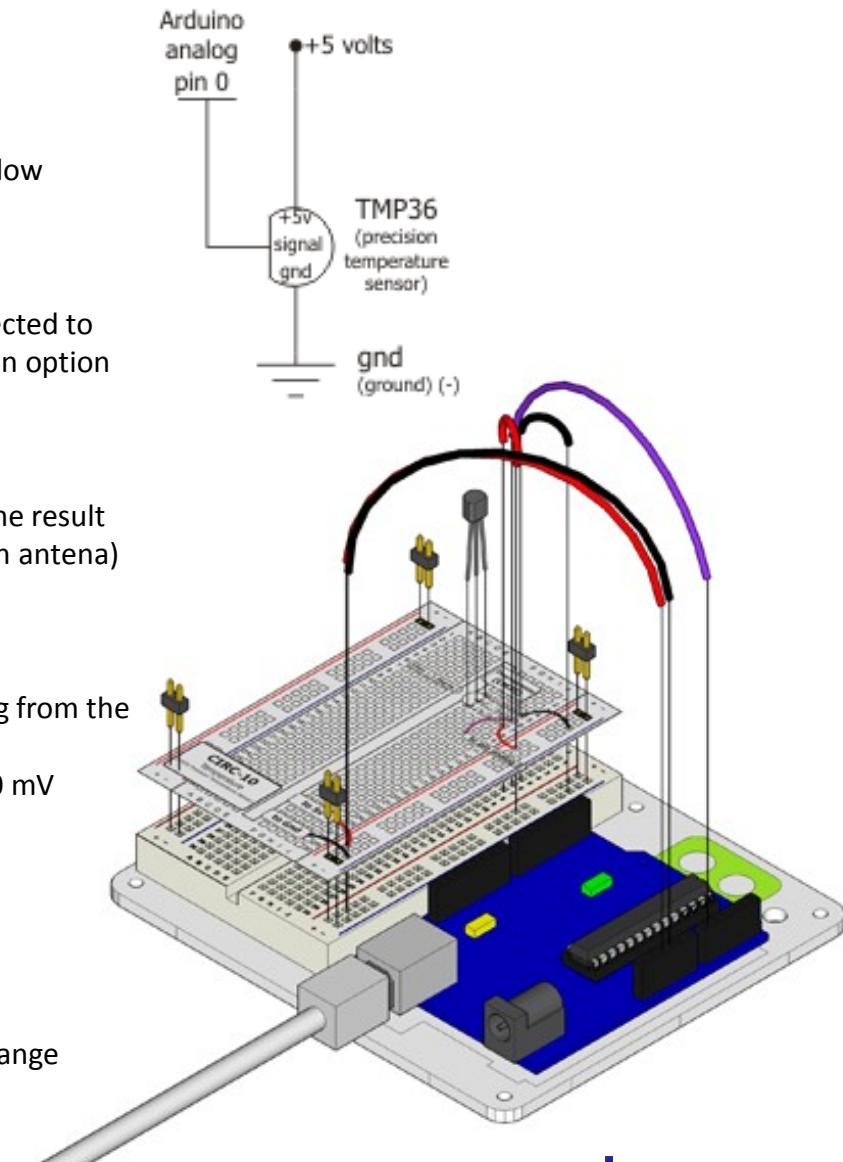
```
/*
 * | Arduino Experimentation Kit Example Code      |
 * | CIRC-10 :: Temperature :: (TMP36 Temperature Sensor) |
 *
 * A simple program to output the current temperature to the IDE's debug window
 * For more details on this circuit: http://tinyurl.com/c89tvd
 */
//TMP36 Pin Variables
int temperaturePin = 0; //the analog pin the TMP36's Vout (sense) pin is connected to
//the resolution is 10 mV /°C (500 mV offset) to make negative temperatures an option
```

```
void setup()
{
    Serial.begin(9600); //Start the serial connection with the computer to view the result
    open the serial monitor last button beneath the file bar (looks like a box with an antenna)
}

void loop()
{
    float temperature = getVoltage(temperaturePin); //getting the voltage reading from the
    temperature sensor
    temperature = (temperature - 0.5) * 100; //converting from 10mV/°C with 500 mV
    //offset to degrees ((voltage - 500mV)*100)
    Serial.println(temperature); //printing the result
    delay(1000); //waiting a second
}

/* getVoltage() – returns the voltage on the analog input defined by pin */
float getVoltage(int pin){
    return (analogRead(pin) * .004882814); //converting from a 0 to 1023 digital range
    // to 0 to 5 volts (each 1 reading equals ~ 5 mV)
}
```

<http://www.oomlout.com/a/products/ardx/circ-10>



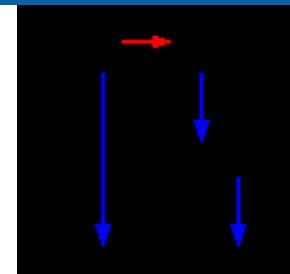
RETOUR

# PONT DIVISEUR

Le diviseur de tension est un montage électronique simple qui permet de diviser une tension d'entrée. Le circuit est par exemple constitué de deux résistances en série:

Une tension  $U$  est appliquée en entrée sur ces deux résistances et la tension de sortie  $U_2$  est mesurée aux bornes de  $R_2$ .

$$U_2 = U \frac{R_2}{R_1 + R_2}$$



**Application:** lecture de tensions plus élevées

Si l'on veut lire des  $V > 5$  volts (e.g. 0 à 24 volts), il vous faudra alors utiliser ce pont diviseur de tension.

Exemple avec un pont diviseur pour  $U_{\max} = 24V$ :

$U_{2\max}$  est fixé à 5V à cause de l'Arduino, fixons arbitrairement  $R_2 = 1k\Omega$ , la valeur nécessaire pour  $R_1$ :  
 $\Rightarrow R_1 = (U/U_2) * R_2 - R_2 = (24/5) * 1000 - 1000 = 3.8 k\Omega$

Ainsi, quand la tension est de 24 volts (le maximum) en tête de pont (tension  $U$ ), la tension maximum sur l'entrée analogique ( $U_2$ ) est de 5V.

**DÉCODAGE DE LA LECTURE ANALOGIQUE** (rappel: 10bits de résolution):

Quelle est la tension à l'entrée du pont si analogRead() retourne la valeur 436?

**Méthode 1: la règle de trois:** analogRead() donne une valeur entre 0 et 1023 (soit 1024 valeurs possible) pour une tension variant entre 0 et 24V  $\Rightarrow (24V/2^{10}) * 436 = 10.22 V$

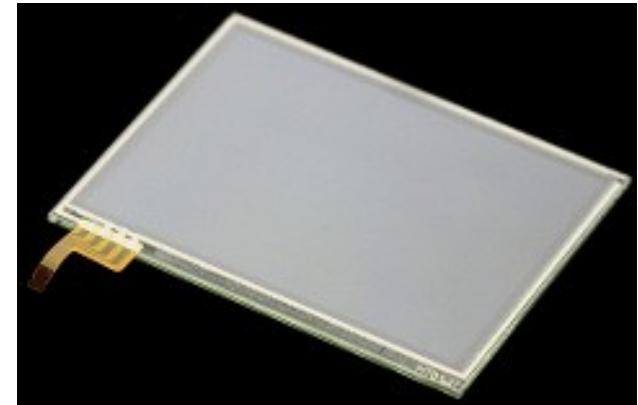
**Méthode 2: utiliser le calcul du pont:** chaque intervalle de analogRead() donne  $5/2^{10} = 4.88mV \Rightarrow U_2 = 436 * 4.88 \times 10^{-3} = 2.12V \Rightarrow U = U_2 * (R_1 + R_2) / R_2 = 2.12 * (3800 + 1000) / 1000 = 10.22 V$

Remarque: en utilisant un pont diviseur, on perd aussi de la précision dans la mesure puisque chaque intervalle représente une variation de tension  $U$  de  $24V / 2^{10} = 23.4 mV$  au lieu de  $4.88mV$ .

RETOUR

Les petits écrans tactiles résistifs à 4-fils sont maintenant très bon marché (<10€), car produits en quantités énormes pour les téléphones mobiles, PDA et console de jeux (Nintendo DS...)

Utiliser des écrans plus grands est aussi possible pour moins de 150€, car la popularité des netbooks avec des écrans de 7"et 10" a fait diminuer leurs prix.



Bien qu'ils soient livrés avec une électronique de commande et une interface USB, ces écrans sont majoritairement en 4 fils résistifs => un Arduino peut piloter un écran tactile de 10" (et plus)

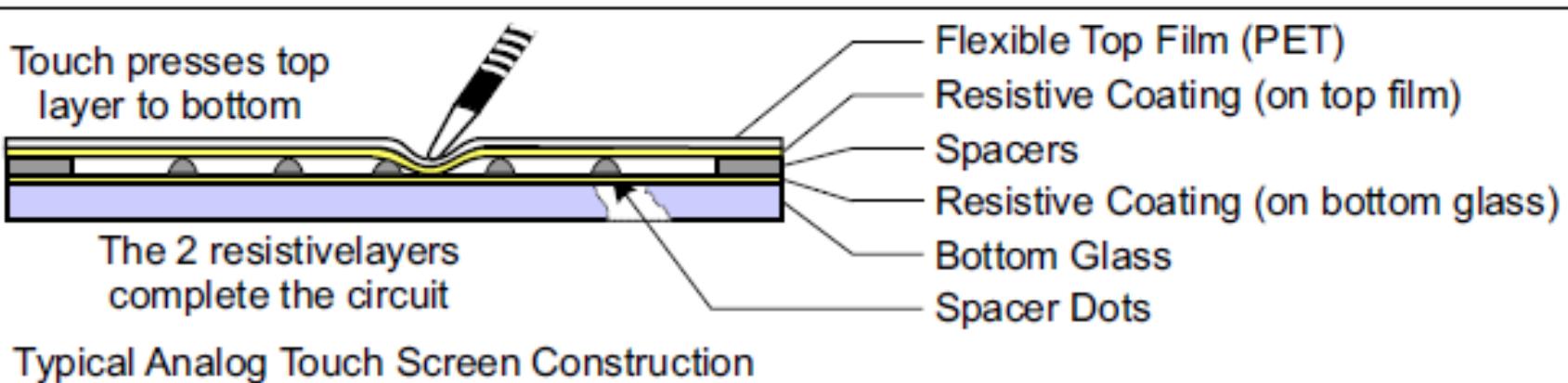
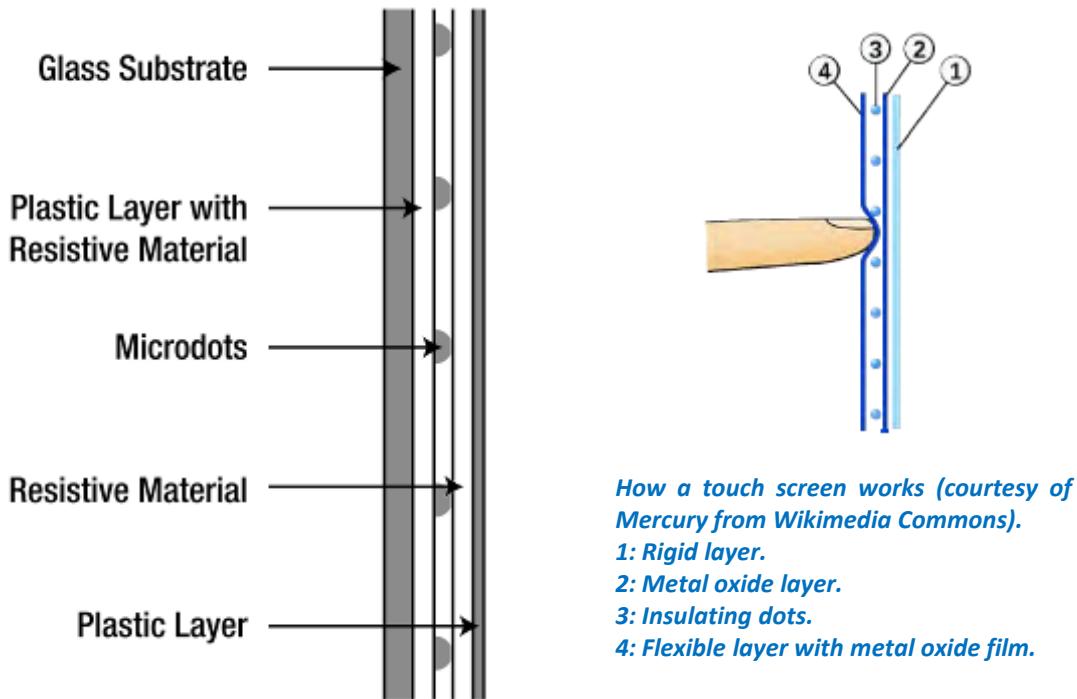
Notez cependant que cet "écran tactile" n'est en réalité que le verre transparent qui s'adapte sur l'avant d'un écran LCD par exemple.

Ils sont très bons pour créer des panneaux de contrôle personnalisés avec les "boutons" imprimés sur une feuille derrière l'écran tactile. L' Arduino capturant les coordonnées (X,Y) et les comparant avec les coordonnées du bouton pressé. Bien sûr, votre panneau de contrôle pourraient représenter quelque chose, pas seulement les boutons.

Vous pourriez avoir un curseur dessus pour sélectionner le volume ou le niveau de température en touchant quelque part le long d'une échelle, ou ce pourrait être un plan d'une maison afin que vous puissiez contrôler les lumières dans différentes pièces en touchant la partie correcte du plan de masse...

Dans ce qui suit, nous allons mettre en œuvre un écran tactile de la Nintendo DS sur une feuille de papier pour déclencher des actions.

# TOUCH PAD DS: COMMENT IL EST FAIT



# TOUCH PAD DS: COMMENT IL FONCTIONNE

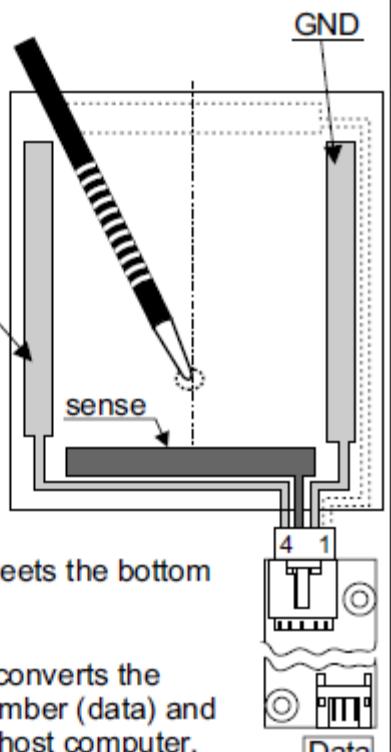
## Capturing the "X" Touch

To get the "X" touch position, the controller sets Pin4 to +5V and Pin2 to GND (0V).

Pin1 is left unconnected.

The controller uses Pin3 to read the voltage where the top layer meets the bottom layer.

The controller converts the voltage to a number (data) and sends it to the host computer.



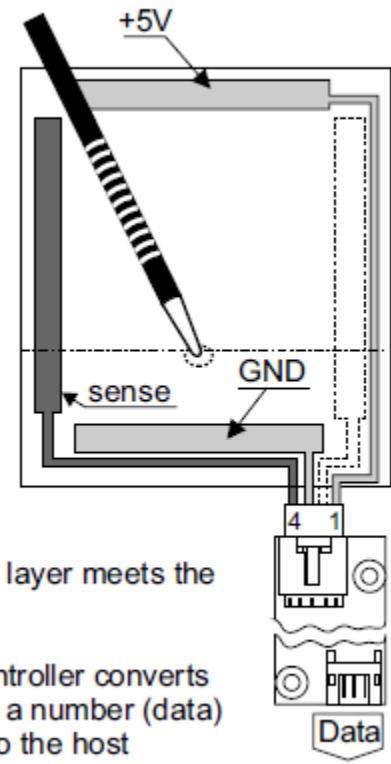
## Capturing the "Y" Touch

To get the "Y" touch position the controller sets Pin1 to +5V and Pin3 to GND (0V)

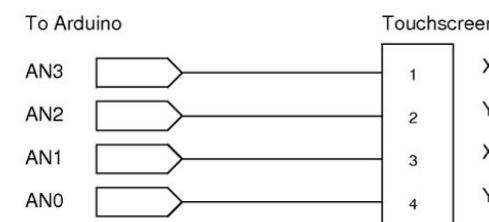
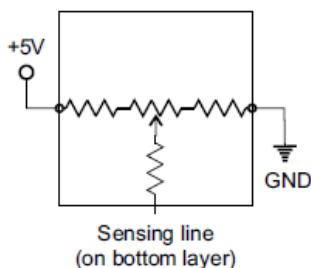
Pin2 is left unconnected.

The controller uses Pin4 to read the voltage where the top layer meets the bottom layer.

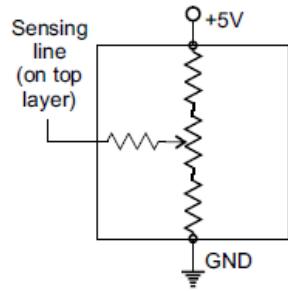
Again, the controller converts the voltage to a number (data) and sends it to the host computer.



Circuit for X position sensing



Circuit for Y position sensing

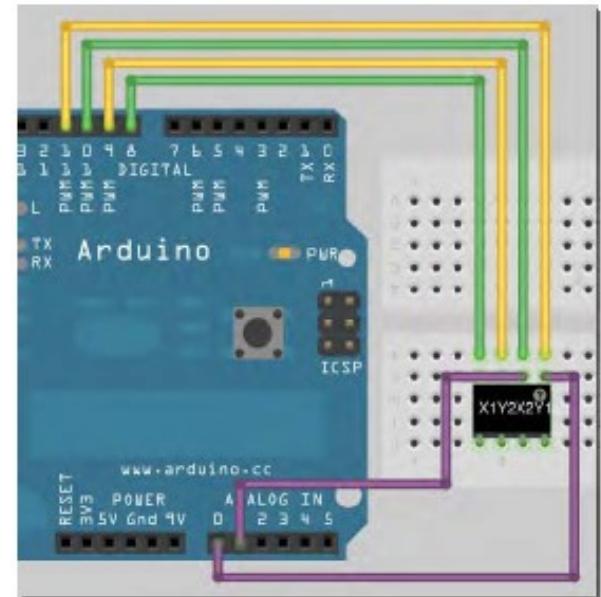


```
// Project 33 – TOUCH SCREEN
// Power connections
#define Left 8 // Left (X1) to digital pin 8
#define Bottom 9 // Bottom (Y2) to digital pin 9
#define Right 10 // Right (X2) to digital pin 10
#define Top 11 // Top (Y1) to digital pin 11
// Analog connections
#define topInput 0 // Top (Y1) to analog pin 0
#define rightInput 1 // Right (X2) to analog pin 1
int coordX = 0, coordY = 0;
void setup() {
Serial.begin(38400);
}

void loop() {
if (touch()) // If screen touched, print co-ordinates
{
Serial.print(coordX);
Serial.print(" ");
Serial.println(coordY);
delay(250);
}
}//fin du void

boolean touch() // return TRUE if touched, and set coordinates to touchX and touchY
{
boolean touch = false;
// get horizontal co-ordinates
pinMode(Left, OUTPUT);
digitalWrite(Left, LOW); // Set Left to Gnd
pinMode(Right, OUTPUT); // Set right to +5v
digitalWrite(Right, HIGH);
pinMode(Top, INPUT); // Top and Bottom to high impedance
pinMode(Bottom, INPUT);
delay(3);
coordX = analogRead(topInput);
// get vertical co-ordinates
pinMode(Bottom, OUTPUT); // set Bottom to Gnd
digitalWrite(Bottom, LOW);
pinMode(Top, OUTPUT); // set Top to +5v
digitalWrite(Top, HIGH);
pinMode(Right, INPUT); // left and right to high impedance
pinMode(Left, INPUT);
delay(3);
coordY = analogRead(rightInput);
// if co-ordinates read are less than 1000 and greater than 0 then the screen has been touched
if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}
return touch;
}
```

*Beginning Arduino Michael McRoberts APRESS*



Arduino	Breakout
Digital Pin 8	X1
Digital Pin 9	Y2
Digital Pin 10	X2
Digital Pin 11	Y1
Analog Pin 0	Y1
Analog Pin 1	X2

```
/**  
 * Touch Control Panel  
 * Copyright 2009 Jonathan Oxer <jon@oxer.com.au>  
 * Copyright 2009 Hugh Blemings <hugh@blemings.org>  
  
• Reads touch coordinates on a Nintendo DS touch screen attached to an  
• Arduino and compares them to defined hot zones representing buttons and sliders.  
• If a touch occurs within a hot zone a matching event message is sent to the host via the serial port.  
• Based on the ReadTouchscreen example included in the TouchScreen library.  
* www.practicalarduino.com/projects/touch-control-panel  
*/  
#include <TouchScreen.h>  
TouchScreen ts(3, 1, 0, 2);  
void setup()  
{  
    Serial.begin(38400);  
}  
void loop()  
{  
    int coords[2];  
    ts.read(coords);  
    Serial.print(coords[0]);  
    Serial.print(",");  
    Serial.print(coords[1]);  
  
    if((coords[0] > 696) && (coords[0] < 866) && (coords[1] > 546) && (coords[1] < 831)) { Serial.print(", Fan ON"); }  
    if((coords[0] > 696) && (coords[0] < 866) && (coords[1] > 208) && (coords[1] < 476)) { Serial.print(", Fan OFF"); }  
    if((coords[0] > 420) && (coords[0] < 577) && (coords[1] > 540) && (coords[1] < 866)) { Serial.print(", Drapes OPEN"); }  
    if((coords[0] > 420) && (coords[0] < 577) && (coords[1] > 208) && (coords[1] < 476)) { Serial.print(", Drapes CLOSE"); }  
    if((coords[0] > 139) && (coords[0] < 327) && (coords[1] > 208) && (coords[1] < 866)) { Serial.print(", Illumination:"); }  
    Serial.print(constrain(map(coords[1], 318, 756, 0, 100), 0, 100));  
    Serial.print("%");  
}  
Serial.println();  
delay (100);  
}
```

Librairie TouchScreen: <https://github.com/practicalarduino/TouchScreen>

<https://github.com/practicalarduino/TouchControlPanel/blob/master/TouchControlPanel.pde>

<http://www.sparkfun.com/tutorials/139>

## Tiré de la datasheet: doc2559 d'ATMEL:

“The ADC accuracy also depends on the ADC clock.

The recommended maximum ADC clock frequency is limited by the internal DAC in the conversion circuitry.

By default, the successive approximation circuitry requires an input clock frequency between **50kHz** and **200kHz** to get maximum resolution (10bits) (a normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry)

However, frequencies up to 1 MHz do not reduce the ADC resolution significantly.

*Operating the ADC with frequencies greater than 1 MHz is not characterized.*

For Arduino and f=16MHz Clock: since the ADC clock needs to be between 50 kHz & 200 kHz for 10 bit accuracy, we can only use the **128 prescaler** and can achieve a **125 kHz ADC clock**.

If a lower resolution than 10 bits is possible, the input clock frequency to the ADC can be higher than 200kHz to get a higher sample rate.

## Tiré de la datasheet ATMEGA328P: doc8271 d'ATMEL

### 24.9.2 ADCSRA – ADC Control and Status Register A

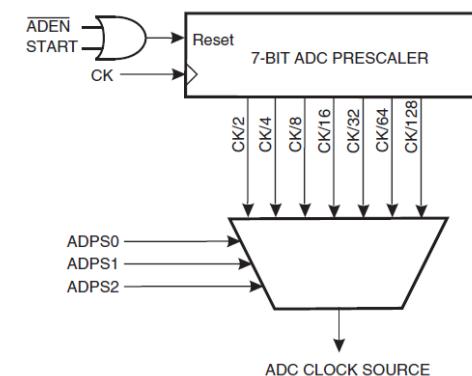
Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### • Bits 2:0 – ADPS[2:0]: ADC Prescaler Select Bits

These bits determine the division factor between the system clock frequency and the input clock to the ADC.

Table 24-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor	f(MHz)/Horloge à 16MHz
0	0	0	2	
0	0	1	2	8
0	1	0	4	4
0	1	1	8	2
1	0	0	16	1
1	0	1	32	0,5
1	1	0	64	0,25
				0,125



Prescaler de la librairie ARDUINO (wiring.c, line 276.) pour garantir les 10bits de résolution car  $50\text{kHz} < (16\text{MHz}/128 = 125\text{kHz}) < 200\text{kHz}$

sketch\_mar23a | Arduino 1.0.2

Fichier Édition Croquis Outils Aide

sketch\_mar23a FastAnalogRead §

```
//TITRE: FASTANALOGREAD
//QUE FAIT CE PROGRAMME:
// IL TESTE LA VITESSE DE LECTURE D'UNE ENTREE ANALOGIQUE
// EN MODIFIANT LES PRESCALERS
// This example code is in the public domain.
// J. Grisolia (2013)

int start;
int i,j;
int FASTADC;
int pin_analogique;
float somme_analogique;

//ADC PRESCALER BITS
// ATTENTION LE PRESCALER PAR DEFAUT EST LE 128, AUGMENTER LA VITESSE DE TRAITEMENT
// DE L'ANALOGREAD SIGNIFIE DIMINUER LE PRESCALER
int prescaler[] = {0, 2, 4, 8, 16, 32, 64, 128};

void setup() {
pin_analogique=0;
somme_analogique=0;

Serial.begin(9600) ;
Serial.println("Test de l'ADC avec différents PRESCALER mesuré sur 1000 appels de la fonction ANALOGREAD: ") ;

//Boucle sur les prescaler
```

```
for (j = 0 ; j < 8 ; j++) {
FASTADC = prescaler[j];
switch (FASTADC) {
case 0: // prescale à 0
cbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 2: // prescale à 2
cbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 4: // prescale à 4
cbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 8: // prescale à 8
cbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 16: // prescale à 16
sbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
case 32: // prescale à 32
sbi(ADCSRA,ADPS2) ;
cbi(ADCSRA,ADPS1) ;
sbi(ADCSRA,ADPS0) ;
break;
case 64: // prescale à 64
sbi(ADCSRA,ADPS2) ;
sbi(ADCSRA,ADPS1) ;
cbi(ADCSRA,ADPS0) ;
break;
default: break;
}

start = millis() ;
for (i = 0 ; i < 1000 ; i++) {
// analogRead(pin_analogique) ;
somme_analogique= somme_analogique + analogRead(pin_analogique);
}//Fin du i

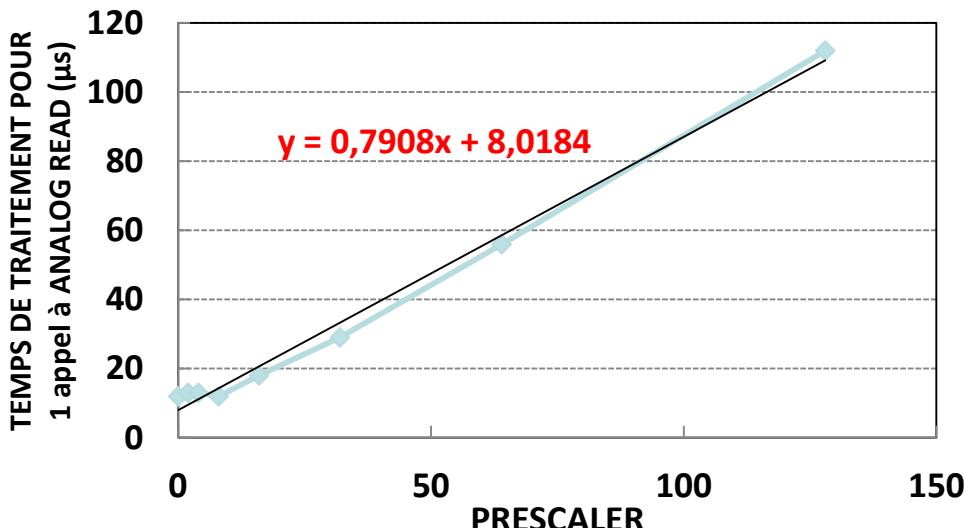
Serial.print("Prescaler ");
Serial.print(FASTADC);
Serial.print(": ");
Serial.print(millis() - start) ;
Serial.print(" msec, ");
Serial.print("T(C): "); //Mesure de température avec le TMP36
//Serial.println((analogRead(pin_analogique)*4.88-500)/10); //Affichage de la température
Serial.println(((somme_analogique/1000)*4.88-500)/10); //Affichage de la température
somme_analogique=0;

}//Fin du j

}//Fin du setup

void loop() {
}
```

**cbi : met à 0 le bit considéré**  
**sbi : met à 1 le bit considéré**



<http://arduino.cc/forum/index.php/topic,6549.0.html>  
<http://www.arduino.cc/en/Reference/PortManipulation>  
<http://playground.arduino.cc/Code/BitMath>  
<http://www.fiz-ix.com/2012/01/how-to-configure-arduino-timer-2-registers-to-drive-an-ultrasonic-transducer-with-a-square-wave/>  
<http://www.marulaberry.co.za/index.php/tutorials/code/arduino-adc/>

On double la fréquence, et on obtient sensiblement le double d'échantillons/s entre deux mesures successives

Prescaler	Fréquence (MHz) - Horloge 16MHz	Temps lecture (1000 appels ANALOGREAD) (ms)	Temps lecture/appel (μs)	Température avec TMP36 sur analog read()	Echantillons par seconde (théorie: f/13 cycles)	Echantillons par seconde (expérience: 1/temps lecture)	Rapport fréquence (théorique)	Rapport fréquence (expérimental)
0		19	19	449,22	0	52632		
2	8	18	18	449,22	615385	55556		0,95
4	4	18	18	24,12	307692	55556	2	1,00
8	2	20	20	23,42	153846	50000	2	1,11
16	1	28	28	23,2				
32	0,5	41	41	23,05				
64	0,25	69	69	22,97				
128	0,125	121	121	22,88				

### Conclusions:

Du prescaler 128 à 4, on passe de 121μs à 18μs par lecture de la fonction analogique ANALOG READ(), soit environ 10x plus vite, sans perdre "trop" de précision (on reste dans les barres d'erreurs du TMP36), même si il y a un léger offset !

On peut monter tranquillement au moins jusqu'à f=1MHz (prescaler 16)

**Attention: surtout éviter le prescaler 0 et 2 qui donnent des valeurs complètement erronées et qui en plus ne font pas gagner en vitesse de lecture !**

## III - 3 APPLICATIONS DIGITAL & ANALOGIQUE

# LCD GENERALITES

Un afficheur LCD est un périphérique d'affichage bon marché.

Il est facile à interfaçer avec Arduino, car ils contiennent leur propre micro-contrôleur (pastille noire à l'arrière) qui gère la logique d'affichage.

Ce µcontrôleur est l'**Hitachi HD 44780** en standard sur la majorité des afficheurs.



Cette standardisation => beaucoup de plateforme µ-controleur dispose d'une librairie de développement toute prête. C'est également le cas d'Arduino avec la librairie LiquidCrystal.

Il est ainsi possible de commander l'afficheur à l'aide de quelques lignes de commande.

La majorité des afficheurs actuels (e.g. LCD 2x16) disposent de deux autres avantages:

1 - l'afficheur est capable d'utiliser 4 bits ou 8 bits pour le bus de données:

la version 4 bits est avantageuse car elle permet d'envoyer des données vers l'afficheur en n'utilisant que 4 pins au lieu de 8. C'est autant de pins récupérées pour la commande d'autres périphériques ☺

2 - la logique de commande (et la librairie) peut se contenter d'utiliser seulement 2 pins au lieu de 3 (en mettant la commande R/W à la masse, si l'on ne désire pas lire la mémoire de l'afficheur (donc ne faire que de l'affichage))

L'afficheur est donc entièrement commandable avec seulement 6 pins !

Outre les lignes de données DB0 à DB7 que l'on pilote à l'aide des 4 ou 8 pins digitales, et qui véhiculent tout à la fois les informations affichées et les commandes envoyées à l'afficheur,

Trois lignes de contrôle sont également nécessaires à son bon fonctionnement:

1 - E ou *Enable* valide l'afficheur quand la pin est au niveau haut. Il peut alors recevoir des commandes ou des caractères à afficher via ses lignes de données, il est insensible à leur état dans le cas contraire.

2 - R/W pour Read/Write indique si l'on veut **Ecrire une donnée dans l'afficheur (R/W à 0)** ou lire ses informations (**R/W à 1**). Quand on veut économiser des entrées/sorties, cette ligne est fréquemment mise à la masse bloquant l'afficheur dans le mode Ecriture (mode d'utilisation principal) => impossible de lire ses registres d'état et sa mémoire de caractères.

3 - RS pour Register Select indique si les lignes de données véhiculent des informations à afficher (**RS à 1**) ou des commandes de gestion de l'afficheur (**RS à 0**). Diverses commandes, comme le déplacement du curseur de droite à gauche ou l'inverse, avec ou sans effacement de caractère, effacement de l'affichage... sont envoyées sur les lignes de données DB0 à DB7 en mettant RS à 0

DIALOGUE AVEC L'AFFICHEUR dans le cas d'une écriture:

- 1 - Mise à 0 de la ligne R/W
- 2 - RS au niveau désiré selon que l'on souhaite envoyer un caractère OU une commande
- 3 - Positionnement du code du caractère OU de la commande sur DB0 à DB7 (mode 8 bits) ou DB4 à DB7 (mode 4 bits)
- 4 - Mise à 1 de la ligne E permettant d'envoyer ces informations
- 5 - Mise à 0 de la ligne E rendant l'afficheur insensible à l'état de DB0 à DB7

# LCD BROCHAGE ET COMMANDE

Le brochage et les spécifications de l'afficheur sont disponibles dans les datasheets respectives mais en général, elles sont comme suit (les pins 1 et 16 sont indiqués sur l'afficheur).

Table 1: Display Control

Pin	Symbol	Description
1	V <sub>SS</sub>	Ground
2	V <sub>DD</sub>	Supply Voltage for Logic
3	V <sub>0</sub>	Supply Voltage for LCD (Contrast)
4	RS	Register Select
5	R/W	Read/Write
6	CE	Chip Enable
15	LED(+)	Anode of LED Backlight
16	LED(-)	Cathode of LED Backlight

Table 2: Parallel Data

Pin	Symbol	Description
7	DB0	*Data bit 0
8	DB1	*Data bit 1
9	DB2	*Data bit 2
10	DB3	*Data bit 3
11	DB4	Data bit 4
12	DB5	Data bit 5
13	DB6	Data bit 6
14	DB7	Data bit 7

\*Note: Not used in 4-bit mode

► Envoie données/commande en 4 bits

On sépare les 4bits de poids forts et les 4 bits de poids faibles.

Les étapes communes sont :

Masquer 4-bits poids faibles

Envoyer au port LCD

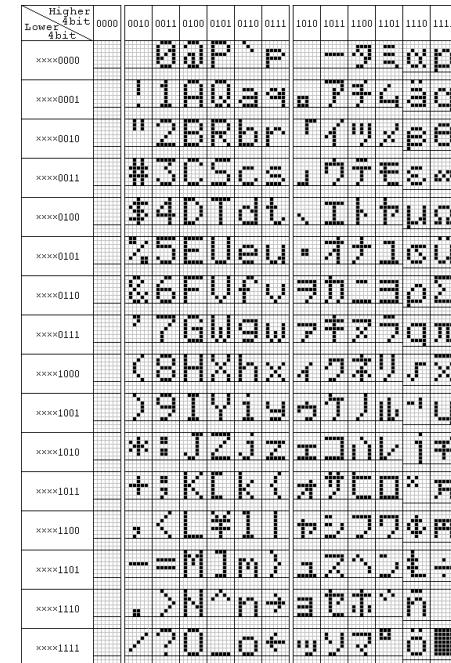
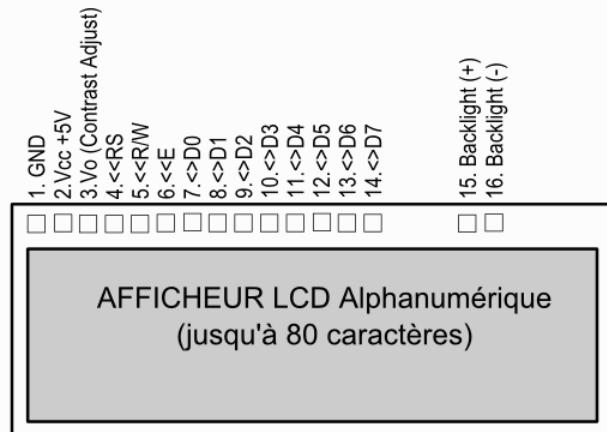
Envoyer ENABLE signal

Masquer 4-bits poids forts

Envoyer au port LCD

Envoyer ENABLE signal

Rem: le mode 4 bits permet de gagner des Pins mais est plus lent que le 8 bits car il faut envoyer les données en 2 fois

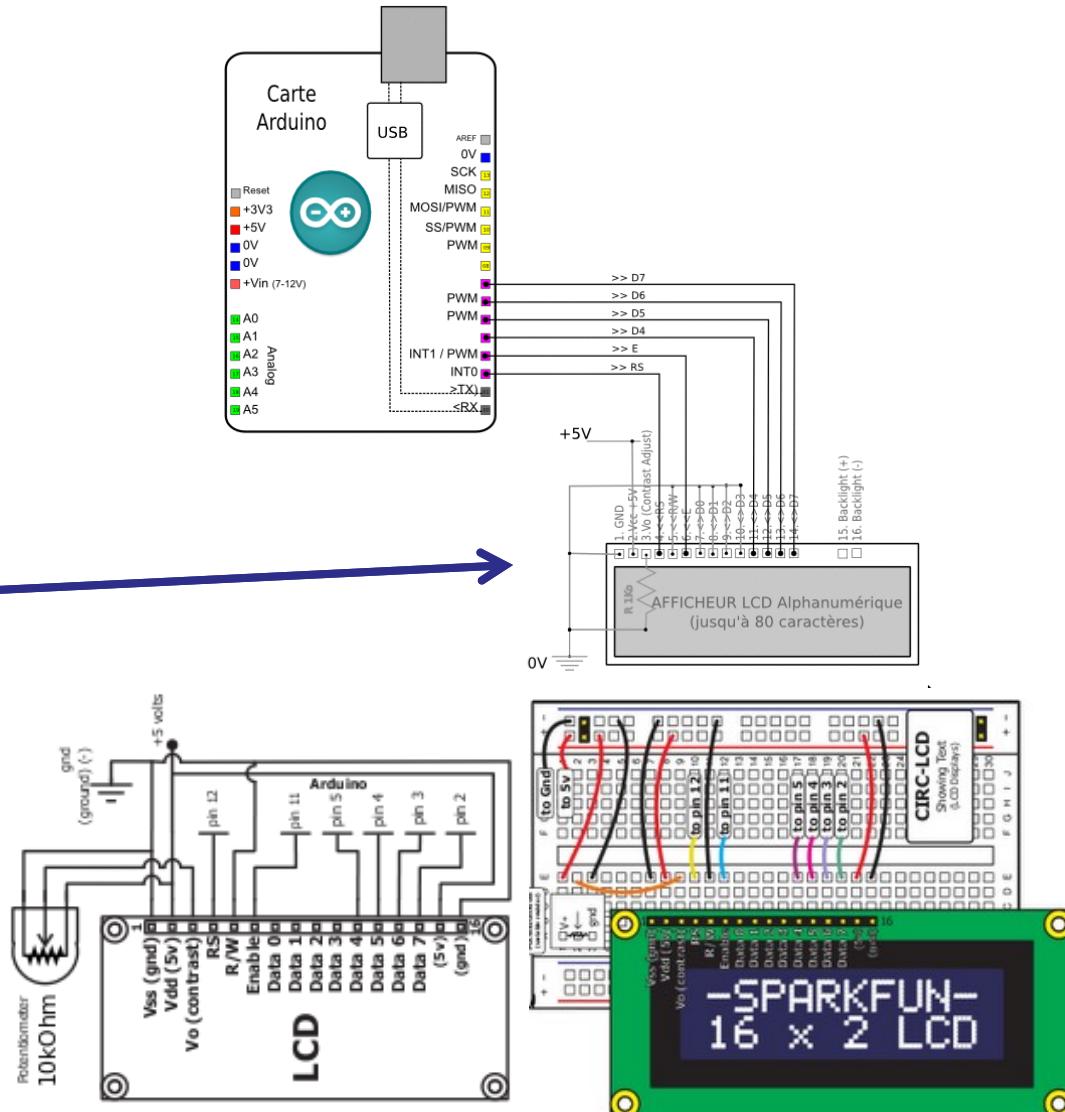


# LCD: EXEMPLE DE CODE

```
// Inclusion de la librairie
#include <LiquidCrystal.h> // Dimension de l'afficheur
const int numRows = 2;
const int numCols = 16; // Initialisation de la librairie
// avec le nbre de pins d'interface
// 4 bit de données dans notre cas.
/* Le montage: Afficheur LCD
* LCD RS - pin 2
* LCD Enable - pin 3
* LCD D4 - pin 4
* LCD D5 - pin 5
* LCD D6 - pin 6
* LCD D7 - pin 7
* LCD R/W - GND
* LCD Vo contrast- potentiomètre 10K (entre Gnd et +5V)
*/
LiquidCrystal lcd(2,3,4,5,6,7);
```

```
void setup()
{
Serial.begin(9600); // Bouton changer thème
lcd.begin(numCols,numRows); //--- Message de bienvenue
lcd.print( "demo LCD" ); // placer curseur sur ligne 2
lcd.setCursor(0,1); // col, row
lcd.print( "Hello!" ); // clignotement curseur plein
lcd.blink();
}

void loop()
{}
```



RETOUR

```
// --- Programme Arduino --- par X. HINAULT - 01/2010
// --- Que fait ce programme ? ---
/* Affiche des messages texte sur l'afficheur LCD*/
// --- Fonctionnalités utilisées ---
// Utilise un afficheur LCD 4x20 en mode 4 bits
// --- Circuit à réaliser ---
// Connexion du LCD sur les broches de la carte Arduino
// Connecter broche RS du LCD sur la broche 2
// Connecter broche E du LCD sur la broche 3
// Connecter broche D4 du LCD sur la broche 4
// Connecter broche D5 du LCD sur la broche 5
// Connecter broche D6 du LCD sur la broche 6
// Connecter broche D7 du LCD sur la broche 7

//***** Entête déclarative *****
// A ce niveau sont déclarées les librairies, les constantes, les variables...
// --- Inclusion des librairies utilisées ---
#include <LiquidCrystal.h> // Inclusion de la librairie pour afficheur LCD
// --- Déclaration des constantes ---
// --- constantes des broches ---
const int RS=2; //declaration constante de broche
const int E=3; //declaration constante de broche
const int D4=4; //declaration constante de broche
const int D5=5; //declaration constante de broche
const int D6=6; //declaration constante de broche
const int D7=7; //declaration constante de broche

// --- Déclaration des variables globales ---
// --- Initialisation des fonctionnalités utilisées ---
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); //initialisation LCD en mode 4 bits

//***** FONCTION SETUP = Code d'initialisation *****
// La fonction setup() est exécutée en premier et 1 seule fois, au démarrage du programme
void setup() { // debut de la fonction setup()

// --- ici instructions à exécuter au démarrage ---
lcd.begin(20,4); // Initialise le LCD avec 20 colonnes x 4 lignes
delay(10); // pause rapide pour laisser temps initialisation
// Test du LCD
lcd.print("LCD OK"); // affiche la chaîne texte - message de test
delay(2000); // pause de 2 secondes
lcd.clear(); // efface écran et met le curseur en haut à gauche
delay(10); // pour laisser temps effacer écran

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

```
***** FONCTION LOOP = Boucle sans fin = cœur du programme *****
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension

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

lcd.print("Arduino..."); // affiche la chaîne texte - message de test
delay(2000); // pause de 2 secondes
lcd.setCursor(9, 1); // 10ème col - 2ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
lcd.print("...pour te"); // affiche la chaîne texte - message de test
delay(2000); // pause de 2 secondes
lcd.setCursor(4, 2); // 5ème col - 3ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
lcd.print("...servir"); // affiche la chaîne texte - message de test
delay(2000); // pause de 2 secondes
lcd.setCursor(12, 3); // 13ème col - 4ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
lcd.print("...amigo"); // affiche la chaîne texte - message de test
delay(2000); // pause de 2 secondes
lcd.clear(); // efface écran et met le curseur en haut à gauche
delay(10); // pour laisser temps effacer écran

// --- ici instructions à exécuter par le programme principal ---
} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin
// *****

// --- Fin programme ---

// ---- memo LCD ---
// LiquidCrystal(rs, enable, d4, d5, d6, d7); // initialisation 4 bits
// lcd.begin(cols, rows); // initialisation nombre colonne/ligne
//
// lcd.clear(); // efface écran et met le curseur en haut à gauche
// lcd.home(); // repositionne le curseur en haut et à gauche SANS effacer écran
//
// lcd.setCursor(col, row); // positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)
// lcd.print("texte"); // affiche la chaîne texte
//
// lcd.cursor(); // affiche la ligne de base du curseur
// lcd.noCursor(); // cache le curseur
// lcd.blink(); // fait clignoter le curseur
// lcd.noBlink(); // stoppe le clignotement du curseur
// lcd.noDisplay(); // éteint le LCD sans modifier affichage
// lcd.display(); // rallume le LCD sans modif affichage
//
// lcd.scrollDisplayLeft(); // décale l'affichage d'une colonne vers la gauche
// lcd.scrollDisplayRight(); // décale l'affichage d'une colonne vers la droite
// lcd.autoscroll(); // les nouveaux caractères poussent les caractères déjà affichés
// noAutoscroll(); // stoppe le mode autoscroll
```

```
/* Mesure de la température à l'aide d'un TMP36 et Affichage sur LCD: MOP-AL162A-BBTW: LCD 2x16
```

Le montage: Afficheur LCD

- \* LCD RS - pin 2
- \* LCD Enable - pin 3
- \* LCD D4 - pin 4
- \* LCD D5 - pin 5
- \* LCD D6 - pin 6
- \* LCD D7 - pin 7
- \* LCD R/W - GND

\* LCD Vo contrast- potentiomètre 10K (entre Gnd et +5V)

Senseur de température:

\* TMP36 Analog Output - Pin A0 (analogique)

// include the library code:

```
#include <LiquidCrystal.h>
```

```
int tempSensorPin = 0; // Pin analogique pour lecture de la tension de sortie du TMP36 (Vout).
```

```
// Resolution: 10 mV / degree celsius avec une offset de 500 mv. // Definition du caractère °
```

// initialize the library with the numbers of the interface pins

```
byte degrees[8] = { B00000, B01000, B10100, B01000, B00000, B00000, B00000, B00000};
```

```
LiquidCrystal lcd(2, 3, 4, 5, 6, 7);
```

**void setup()**

```
lcd.begin(16, 2); // set up the LCD's number of columns and rows:
```

```
lcd.clear(); //efface l'écran
```

```
lcd.createChar(0, degrees); } // initialiser le caractère ° dans le LCD
```

```
int lastTemp = -100; // mémorise la dernière température affichée
```

**void loop()**

```
float temp = lectureTemp(); // rafraîchit le LCD que si la // température a varié sensiblement
```

```
if (abs(temp-lastTemp)<0.20) return; lastTemp = temp; // Afficher la valeur en évitant le // lcd.clear(), pour éviter l'effet de // scintillement.
```

```
lcd.setCursor(0,0);
```

```
lcd.print( temp );
```

```
lcd.write( 0 ); // affiche le signe degré
```

```
lcd.print( "c" ); // Efface les derniers caractères si // la température chute subitement
```

```
lcd.print( " " ); // ne pas rafraîchir trop souvent
```

```
delay(800); }
```

//Description: // Lecture de la température sur la pin A0 // //Returns: // La température en degré Celcius. //

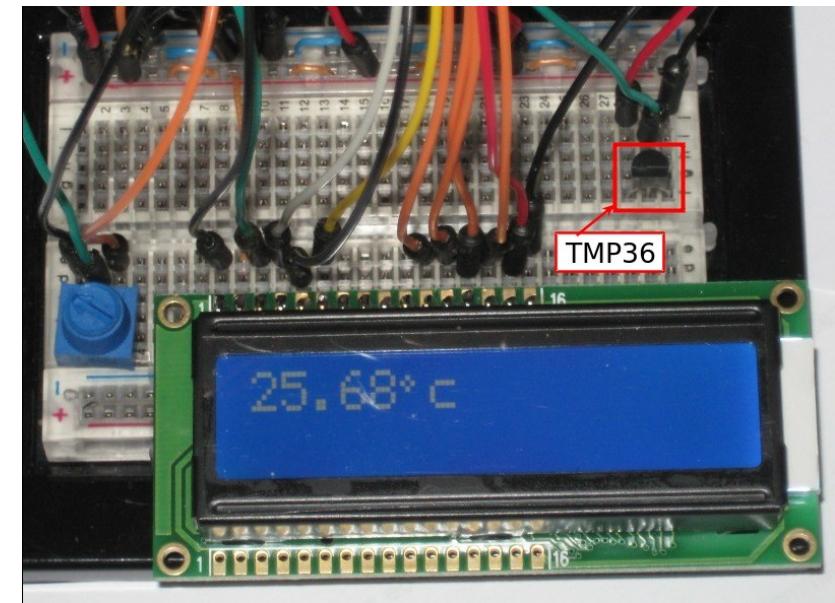
```
float lectureTemp(){ // Lecture de la valeur sur l'entrée analogique // Retourne une valeur entre 0->1024 pour 0->5v
```

```
int valeur = analogRead(tempSensorPin); // Converti la lecture en tension
```

```
float tension = valeur * 5.0; tension /= 1024.0; // Convertir la tension (mv) en température
```

```
float temperature = ((tension * 1000) - 500) / 10;
```

```
return temperature; }
```



**Source:** <http://arduino103.blogspot.com/search/label/sensor>

**Editeur de caractère LCD:** <http://icontexto.com/charactercreator/>

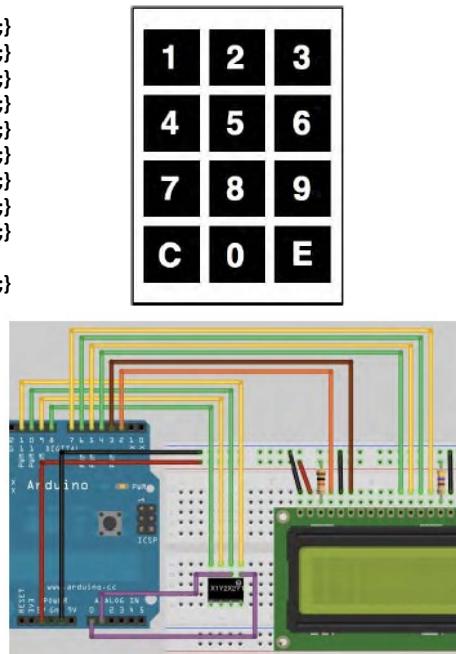
 RETOUR

# TOUCH SCREEN & LCD

```
// Project 34
#include <LiquidCrystal.h>
LiquidCrystal lcd(2, 3, 4, 5, 6, 7); // create an lcd object
//and assign the pins
// Power connections
#define Left 8 // Left (X1) to digital pin 8
#define Bottom 9 // Bottom (Y2) to digital pin 9
#define Right 10 // Right (X2) to digital pin 10
#define Top 11 // Top (Y1) to digital pin 11
// Analog connections
#define topInput 0 // Top (Y1) to analog pin 0
#define rightInput 1 // Right (X2) to analog pin 1
int coordX = 0, coordY = 0;
char buffer[16];
void setup()
{
lcd.begin(16, 2); // Set the display to 16 columns and 2 rows
lcd.clear();
}
void loop()
{
if (touch())
{
if ((coordX>110 && coordX<300) && (coordY>170 && coordY<360)) {lcd.print("3");}
if ((coordX>110 && coordX<300) && (coordY>410 && coordY<610)) {lcd.print("2");}
if ((coordX>110 && coordX<300) && (coordY>640 && coordY<860)) {lcd.print("1");}
if ((coordX>330 && coordX<470) && (coordY>170 && coordY<360)) {lcd.print("6");}
if ((coordX>330 && coordX<470) && (coordY>410 && coordY<610)) {lcd.print("5");}
if ((coordX>330 && coordX<470) && (coordY>640 && coordY<860)) {lcd.print("4");}
if ((coordX>490 && coordX<710) && (coordY>170 && coordY<360)) {lcd.print("9");}
if ((coordX>490 && coordX<710) && (coordY>410 && coordY<610)) {lcd.print("8");}
if ((coordX>490 && coordX<710) && (coordY>640 && coordY<860)) {lcd.print("7");}
if ((coordX>760 && coordX<940) && (coordY>170 && coordY<360)) {scrollLCD();}
if ((coordX>760 && coordX<940) && (coordY>410 && coordY<610)) {lcd.print("0");}
if ((coordX>760 && coordX<940) && (coordY>640 && coordY<860)) {lcd.clear();}
delay(250);
}
}
```

Arduino	Other	Matrix
Digital 2		Enable
Digital 3		RS (Register Select)
Digital 4		DB4 (Data Pin 4)
Digital 5		DB5 (Data Pin 5)
Digital 6		DB6 (Data Pin 6)
Digital 7		DB7 (Data Pin 7)
Gnd		Vss (GND)
Gnd		R/W (Read/Write)
+5v		Vdd
	+5v via resistor	Vo (Contrast)
	+5v via resistor	A/Vee (Power for LED)
	Gnd	Gnd for LED

```
// return TRUE if touched, and set coordinates to touchX and touchY
boolean touch()
{
boolean touch = false;
// get horizontal co-ordinates
pinMode(Left, OUTPUT);
digitalWrite(Left, LOW); // Set Left to Gnd
pinMode(Right, OUTPUT); // Set right to +5v
digitalWrite(Right, HIGH);
pinMode(Top, INPUT); // Top and Bottom to high impedance
pinMode(Bottom, INPUT);
delay(3); // short delay
coordX = analogRead(topInput);
// get vertical co-ordinates
pinMode(Bottom, OUTPUT); // set Bottom to Gnd
digitalWrite(Bottom, LOW);
pinMode(Top, OUTPUT); // set Top to +5v
digitalWrite(Top, HIGH);
pinMode(Right, INPUT); // left and right to high impedance
pinMode(Left, INPUT);
delay(3); // short delay
coordY = analogRead(rightInput);
// if co-ordinates read are less than 1000 and greater than 0 then the
screen has been touched
if(coordX < 1000 && coordX > 0 && coordY < 1000 && coordY > 0) {touch = true;}
return touch;
}
void scrollLCD() {
for (int scrollNum=0; scrollNum<16; scrollNum++) {
lcd.scrollDisplayLeft();
delay(100);
}
lcd.clear();
}
```



## III - 4 FAIRE DE « L'ANALOGIQUE » AVEC DU DIGITAL

# PULSE WIDTH MODULATION (PWM)

Les Arduino ne disposent pas d'un convertisseur **DIGITAL/ANALOGIQUE** au sens strict du terme.  
Impossible d'avoir sur une des connecteurs une tension continue qui soit l'image d'une donnée numérique.  
Par contre, ils sont capables de générer des signaux à modulation de largeur d'impulsion (**PWM** en Anglais: pulse width modulation) qui comblent les manquements du contrôle analogique.

Mais solution élégante car on fait travailler des transistors en commutation (régime bloqué et saturé; cf COURS DISPOSITIFS A SEMICONDUCTEURS 3A IMACS) => sa dissipation est en théorie nulle, puisque soit le I soit le U sont nuls alternativement.

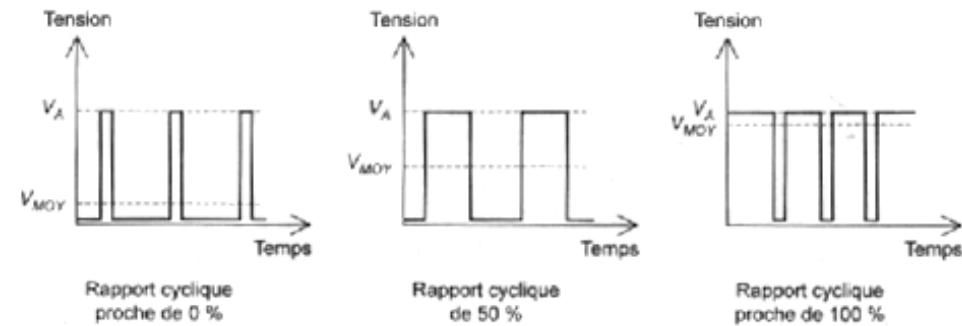
## Les limitations du contrôle analogique:

Si l'on veut contrôler la luminosité d'une LED ou encore la vitesse d'un moteur DC, appliquer une tension plus faible ou plus importante (contrôle analogique) n'implique une obtention des résultats voulus => luminosité plus faible pour une led n'est absolument pas garanti (car c'est une diode (redresseur)).  
De même, le démarrage plus lent du moteur pas assurés (à cause de l'inertie).

## Cycle de service:

Rapport cyclique = le ratio entre la durée du phénomène sur une période et la durée de cette même période ( $\alpha=t_{ON}/T$ )

La fréquence du signal PWM est d'environ ~490Hertz (soit un cycle de service de +/- 2 ms) et ~1kHz selon les pins PWM.



⇒ Duty cycle 100% => signal HAUT | Duty cycle 0% => signal BAS  
⇒ Toutes les valeurs intermédiaires sont évidemment permises

⇒ Même si le signal n'est nullement une tension analogique, il peut le devenir si on lui applique un filtre passe bas (qui ne conserve que la moyenne du signal;  $V_Moy=V_A \cdot R_{Cyc}$ )

## Activation du mode PWM

Attention, il n'y a **pas d'instruction particulière pour activer le mode PWM et il ne faut pas initialiser la pin comme Output avant d'utiliser l'instruction analogWrite()**.

Ensuite, on sélectionne une pin **PWM (3, 5, 6, 9, 10, 11)** au lieu d'une pin analogique.

## *Limite des pins 5 & 6 en PWM:*

*Attention: suite à une contrainte technique (partage de la fonction millis et delays), le contrôle PWM sur les pins 5 et 6 est inconstant pour les valeurs analogWrite de 0 à 10.*

**En conséquence, une valeur 0 sur les pins 5 et 6 n'assure pas forcément un output à LOW!**

## Note de calcul:

La valeur passée à analogWrite() indique quelle est la proportion du « duty cycle » durant lequel la sortie est activée (voir graphique).

Comme la valeur de analogWrite ne peut varier que de 0 à 255:

- Un duty cycle de **50% du temps** donnera: 50% de 255, soit  $255 \times 0.5 = 127.5$  (donc **127 pour l'instruction analogWrite**).

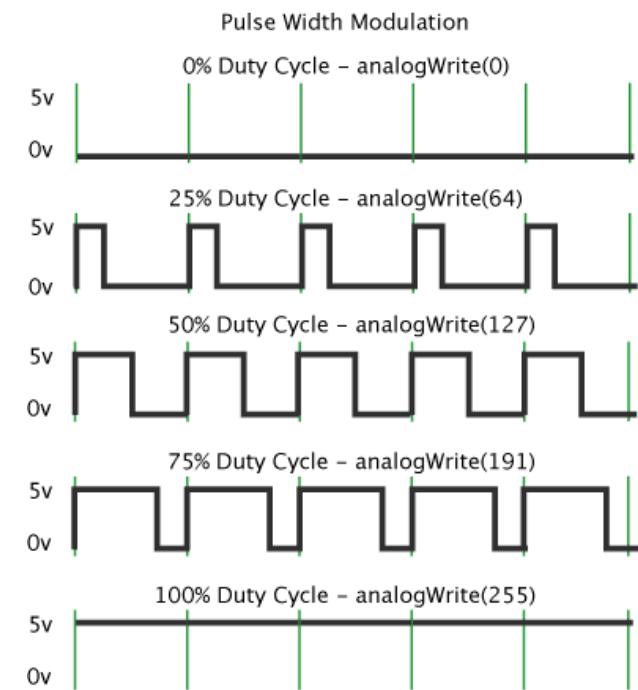
- Ou plus généralement:

**un duty cycle de x % donnera un n de  $(255 \cdot (100/x))$ .**

## OU inversement:

si l'on a une valeur analogWrite() = n,

**le duty cycle sera de  $(n/255) \times 100\%$  de la période.**



Raffinement: filtre passe bas => <http://arduino-info.wikispaces.com/Analog-Output>

Avec « l'astuce » des signaux PWM, on arrive à faire du pseudo analogique,

Mais comment faire si je veux un signal complètement analogique à partir d'un signal numérique ?

**Solution:** mettre un filtre RC en sortie de la PWM

1 - Si le signal de la PWM passe à 1, le condensateur se charge.

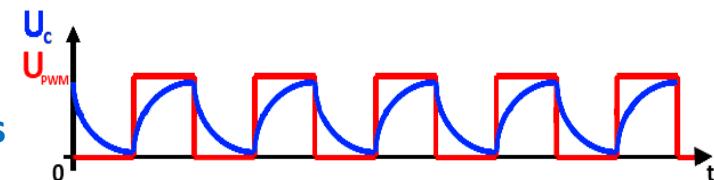
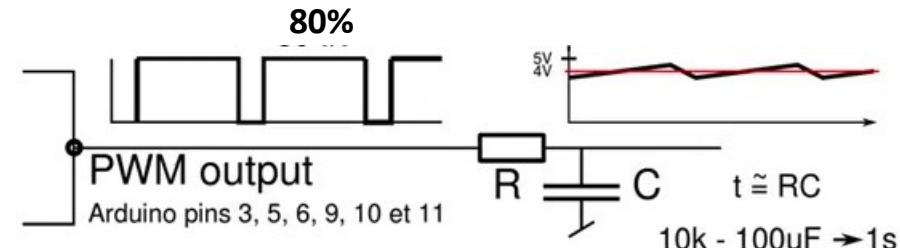
2 - Si le signal de la PWM passe à 0, le condensateur se décharge.  
et ainsi de suite... impliquant une variation de tension aux bornes du condensateur semblable à celle-ci :

Qu'y a-t-il de nouveau par rapport au signal carré ?

Rien de plus, la valeur moyenne du signal bleu est la même que le signal rouge.

*Précisons que dans ce cas, le temps de charge/décharge du condensateur est choisi égal à une demi-période du signal*

Que se passera-t-il si on choisit un temps de charge/décharge plus petit ou plus grand ? Par exemple un RC plus grand :



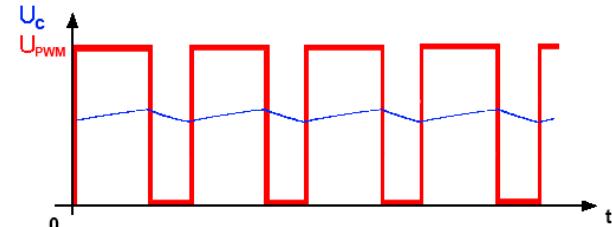
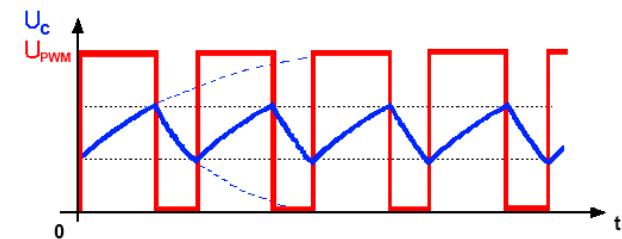
La tension aux bornes du condensateur n'atteint plus le +5V et le 0V comme au chronogramme précédent.

Le couple RC étant plus grand que précédemment, le condensateur met plus de temps à se charger et comme le signal "va plus vite" que le condensateur, il ne peut se charger/décharger complètement.

Que se passera-t-il si on continue à augmenter la constante RC ?

Ce signal s'approche de plus en plus de la valeur moyenne du signal de la PWM !!

A-t-on pour autant gagné la partie ?



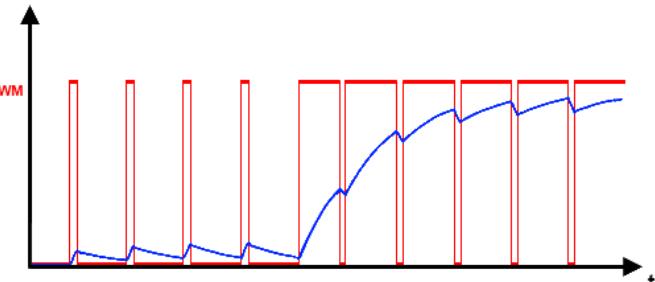
# « VRAIE » SORTIE ANALOGIQUE !

Que faire si l'on veut avoir une belle droite ?

Augmenter encore la constante de temps RC ?

Oui mais attention, il faut prendre des précautions car des problèmes peuvent apparaître:

=> Problème du temps de stabilisation entre deux paliers (rapports cycliques différents)



Plus on augmente le temps de charge avec RC, plus le condensateur mettra du temps pour se stabiliser au palier voulu: or si l'on veut créer un signal analogique qui varie assez rapidement => problème.

Ce qui peut parfois vous sauver c'est la persistance rétinienne des yeux pour une application LED ou qu'un moteur possède de l'inertie !

Et puis, on ne peut à l'inverse diminuer le RC car changer de palier sera plus rapide, mais la tension  $U_c$  aura tendance à suivre le signal: c'est le premier chronogramme que l'on a vu plus haut.

Comment calibrer correctement la constante RC ? Trouver le juste milieu !!!

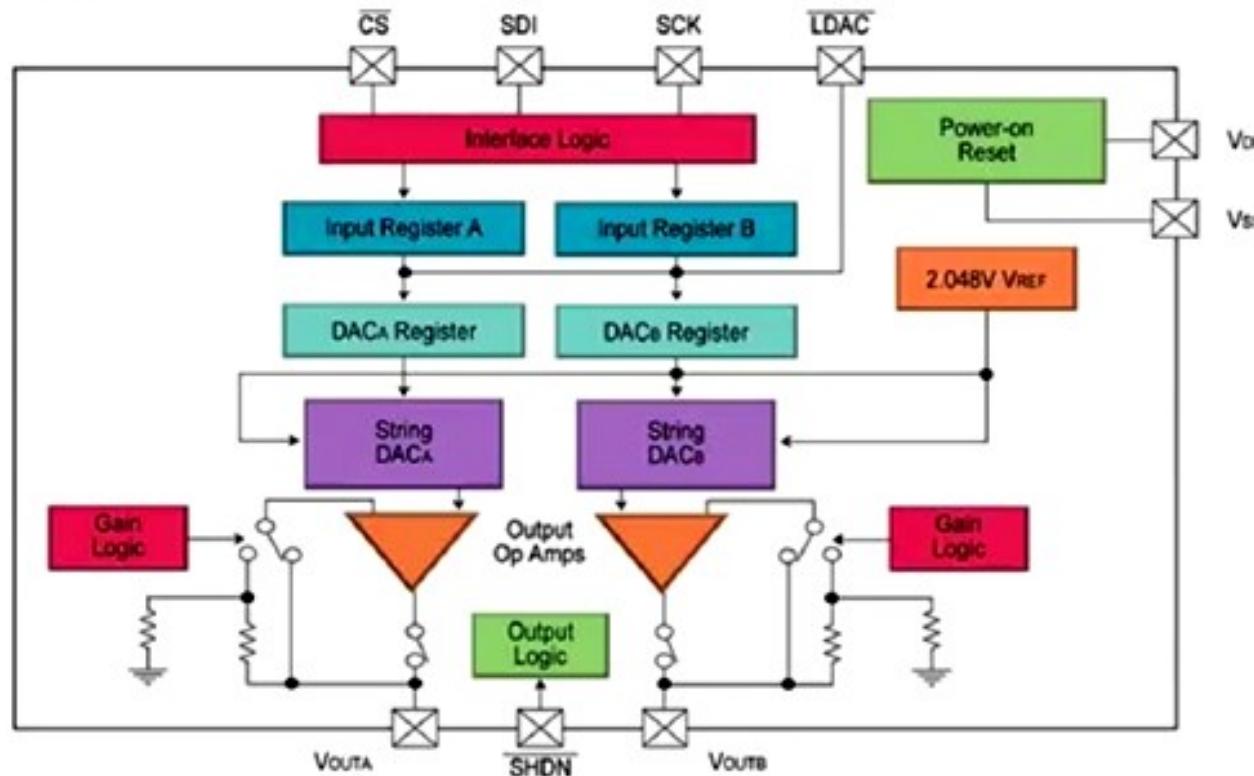
- Si l'on veut un signal qui soit le plus proche possible de la valeur moyenne: constante RC très grande.
- Si au contraire on veut un signal qui soit le plus rapide et que la valeur moyenne soit une approximation, alors il faut que la constante de temps faible.
- Si on veut un signal rapide et le plus proche possible de la valeur moyenne, on peut aussi changer la fréquence de la PWM

# POUR ALLER PLUS LOIN

Exemple de « vrai » convertisseur DIGITAL/ANALOGIQUE :



MCP48X2 Digital-to-Analog  
Converters (DACs)



La valeur PWM est définie sur 8/16 bits lors de l'appel à `analogWrite()`: `analogWrite(myPWMPin, 128);`

Cette valeur est alors comparée à la valeur d'un des registres compteurs `OCRnX` (les 8-bits ou le 16-bits):

**1 - Lorsque le compteur est inférieur à la valeur PWM: la broche de sortie un HAUT;**

**2 - Lorsque le compteur est supérieur à la valeur PWM, la broche de sortie un BAS.**

Avec le timer 8 bit, L'exemple ci-dessus génère un signal carrée car la pin est la même quantité de temps au niveau HAUT (de 0 à 127) puis au niveau BAS (de 128 jusqu'à 255).

Avec le timer 16 bits => il compte jusqu'à  $2^{16}=65536$

Sur les ATMega: il y a trois types de registres timer/counter : **TCCR0B, TCCR1B et TCCR2B.**

**1 - Chaque timer a deux Output Compare Register `OCRnA & OCRnB` => 3timers \*2 OCR = 6 canaux PWM**

**2 – Chaque timer a un prescaler utilisé pour générer une fréquence = fréquence d'horloge/prescaler**

Ce prescaler est défini par 3-bits mémorisés dans les trois bits de poids faibles du registre de timer/counter: **CS02, CS01, CS00.**

Avec ces trois seuls registres (définissant les 3 différents prescalers), les six pins PWM ne peuvent qu'être groupées en trois paires, chaque paire ayant son prescaler propre et donc sa fréquence propre:

**Pin 5 et 6: TCCR0B (timer0/8bits), Pin 9 et 10: TCCR1B (timer1/16bits), Pin 3 et 11: TCCR2B (timer2/ 8bits)**

Par défaut sur l'Arduino Diecimila: Pins 5 et 6: 1kHz et Pins 9, 10, 11 et 3: 500Hz (prescaler 64)

⇒ La fréquence du signal PWM est donc déterminée par la valeur du prescaler, la vitesse d'horloge et la résolution du timer (255 ou 65536 pour le timer 8-bit ou 16-bit respectivement).

Il existe deux modes de fonctionnement des timers pour les PWM.

1 – Fast PWM et

2 – Phase-correct PWM.

## FAST PWM :

The following code fragment sets up fast PWM on pins 3 and 11 (Timer 2), using OCR2A as the top value for the timer.

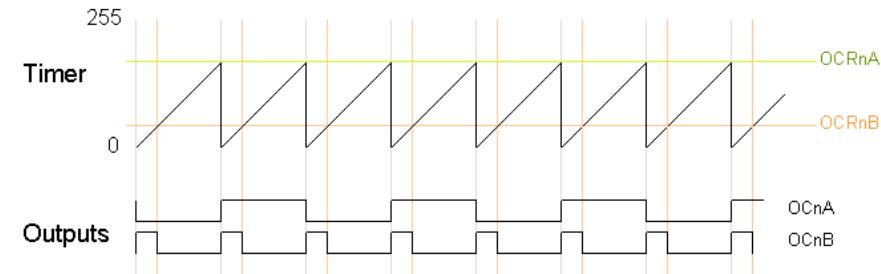
The waveform generation mode bits WGM are set to 111 for fast PWM with OCRA controlling the top limit.

The OCR2A top limit is arbitrarily set to 180, and the OCR2B compare register is arbitrarily set to 50. OCR2A's mode is set to "Toggle on Compare Match" by setting the COM2A bits to 01.

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(WGM22) | _BV(CS22); OCR2A = 180; OCR2B = 50;
```

On the Arduino Duemilanove, these values yield:

- Output A frequency:  $16 \text{ MHz} / 64 / (180+1) / 2 = 690.6\text{Hz}$
- Output A duty cycle: 50%
- Output B frequency:  $16 \text{ MHz} / 64 / (180+1) = 1381.2\text{Hz}$
- Output B duty cycle:  $(50+1) / (180+1) = 28.2\%$



## PHASE CORRECT PWM :

The following code fragment sets up phase-correct PWM on pins 3 and 11 (Timer 2), using OCR2A as the top value for the timer.

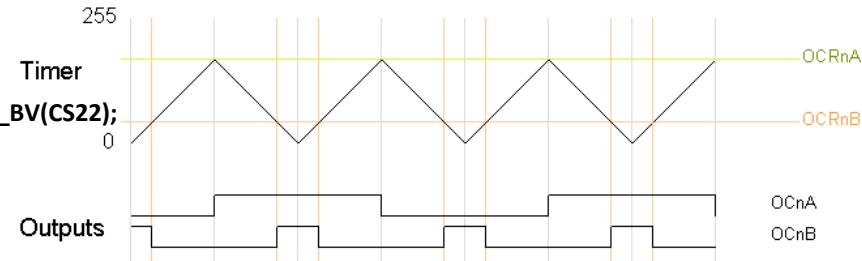
The waveform generation mode bits WGM are set to 101 for phase-correct PWM with OCRA controlling the top limit.

The OCR2A top limit is arbitrarily set to 180, and the OCR2B compare register is arbitrarily set to 50. OCR2A's mode is set to "Toggle on Compare Match" by setting the COM2A bits to 01.

```
pinMode(3, OUTPUT);
pinMode(11, OUTPUT);
TCCR2A = _BV(COM2A0) | _BV(COM2B1) | _BV(WGM20); TCCR2B = _BV(WGM22) | _BV(CS22);
OCR2A = 180; OCR2B = 50;
```

On the Arduino Duemilanove, these values yield:

- Output A frequency:  $16 \text{ MHz} / 64 / 180 / 2 / 2 = 347.2\text{Hz}$
- Output A duty cycle: 50%
- Output B frequency:  $16 \text{ MHz} / 64 / 180 / 2 = 694.4\text{Hz}$
- Output B duty cycle:  $50 / 180 = 27.8\%$



<http://www.righto.com/2009/07/secrets-of-arduino-pwm.html>

# CHANGER LA FREQUENCE PWM - 2

Les registres TCCRnA et B ont différents bits:

- Waveform Generation Mode bits – WGM
- Clock Select bits – CS
- Compare Match Output bits – COMnA & COMnB

Table 18-8. Waveform Generation Mode Bit Description

	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on <sup>(1)(2)</sup>
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	—	—	—
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	—	—	—
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Notes:  
1. MAX=0xFF  
2. BOTTOM=0x00

Dans la partie setup () de votre code Arduino:

1 – Choisir le mode (Fast PWM...) dans les registres TCCRnA pertinents

## 18.11.1 TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	TCCR2A
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	—	—	WGM21	WGM20	
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

2 - Définir ou effacer les bits CS22, CS21, CS20 des registres TCCRnB pertinents pour définir les prescaler.

## 17.11.2 TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	TCCR2B
(0xB1)	FOC2A	FOC2B	—	—	WGM22	CS22	CS21	CS20	
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Les fréquences peuvent être alors définies comme suit suivant le Timer utilisé:

8-bit

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

16-bit

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

The counter counts from BOTTOM to TOP then restarts from BOTTOM.

The N variable represents the prescale factor (1, 8, 64, 256, or 1024).

<http://www.marulaberry.co.za/index.php/tutorials/code/pulse-width-modulation/>

```
void setup() {
    //setup OC2A (pin 11) as output
    pinMode(11, OUTPUT);

    // Setup the Timer/Counter Control Registers - TCCR2A
    // - set WGM bits to 011 - Fast PWM mode
    // - set COM2A CS2 bits to 10 - Non Inverted PWM
    TCCR2A = (1 << COM2A1) | (1 << WGM21) | (1 << WGM20);

}

void loop() {
    int duty = 0;
    OCR2A = 0; // set initial duty cycle
    // Change the duty cycle every 3 seconds
    while(1) {
        if (duty > 255) duty = 0;
        // set the Output Compare Register (channel A)
        // to what ever duty cycle we want
        OCR2A = duty;
        // We can block the main loop as long as we want and
        // PWM is still going on in the background at whatever
        // duty cycle we set before we called delay
        delay(3000);

        duty += 50; // increase duty cycle
    }
}
```

## EXEMPLE:

Si vous souhaitez configurer les broches 5 et 6 pour produire un signal PWM à la fréquence la plus élevée possible, i.e. 64kHz => TCCR0B = xxxxx001 (timer 0 8 bit)

CS02	CS01	CS00	DESCRIPTION	Fréquence (pins 5 et 6 (OC0A and OC0B): TIMERO (8BIT PWM))	Fréquence [(pins 9, 10, (OC1A, OC1B) TIMER1 (16BIT PWM)] ET [pins 3, 11 (OC2A, OC2B): TIMER2 (8BIT PWM)]
0	0	0	Timer/Counter Disabled		
0	0	1	No Prescaling	64kHz	32kHz
0	1	0	Clock / 8	8kHz	4kHz
0	1	1	Clock / 64	1kHz (défaut)	500Hz (défaut)
1	0	0	Clock / 256	250Hz	125Hz
1	0	1	Clock / 1024	62.5Hz	31.25Hz

### Exemple:

fréquence d'horloge à  $f=16MHz$

Prescaler à 64

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

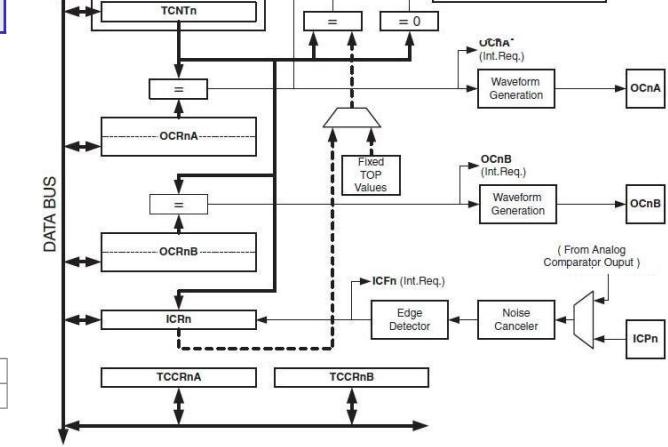
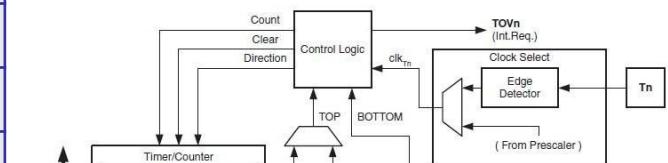
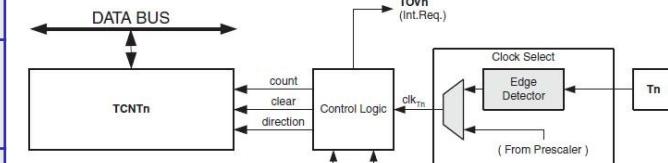
$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

Attention par défaut dans wiring.c: Timer 0 is initialized to Fast PWM,  
While Timer 1 and Timer 2 is initialized to Phase Correct PWM.

TCCR1B	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
TCCR0B	FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

Timer/Counter Control Register 0 B



//Premièrement, effacer les bits du prescaler

```
int prescalerVal = 0x07; //cela crée une variable appelée prescalerVal qui est égale au binaire "00000111"
```

```
TCCR0B &= ~prescalerVal; // entre dans le registre TCCR0B l'opération ET entre la valeur de TCCR0B et l'inverse (tilde) du binaire "00000111" soit "11111000"
```

//Ensuite, mets les bonnes valeurs de bits du prescaler

```
int prescalerVal = 1; //mets la prescalerVal égale au binaire "00000001"
```

```
TCCR0B |= prescalerVal; //réalise un OU entre la valeur précédente de TCCR0B "11111000" et le binaire "00000001"
```

⇒ Résultat "11111001" CS02, CS01 ont été effacés et CS00 activé !

⇒ Aucun prescaling => fréquence maximale

## AUTRE MÉTHODE:

//Premièrement, effacer les bits du prescaler

```
// fait 1<<2 => « 00000001 » => « 00000100 »
```

```
// ~ « 00000100 » = « 11111011 »
```

```
// TCCR0B ET « 11111011 »
```

```
TCCR0B &= ~(1<<CS02);
```

```
//idem pour CS01, // fait 1<<1 => « 00000001 » => « 00000010 »
```

```
TCCR0B &= ~(1<<CS01);
```

//Mets le CS00 à la bonne valeur

```
TCCR0B |= (1<<CS00); //OU avec le TCCR0B.
```

<http://www.atmel.com/Images/doc2505.pdf>

<http://www.avrfreaks.net/>

<https://sites.google.com/site/qeewiki/books/avr-guide/timers-on-the-atmega328>

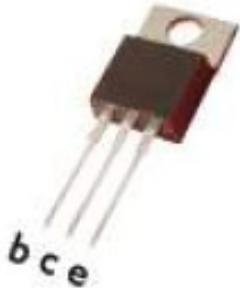
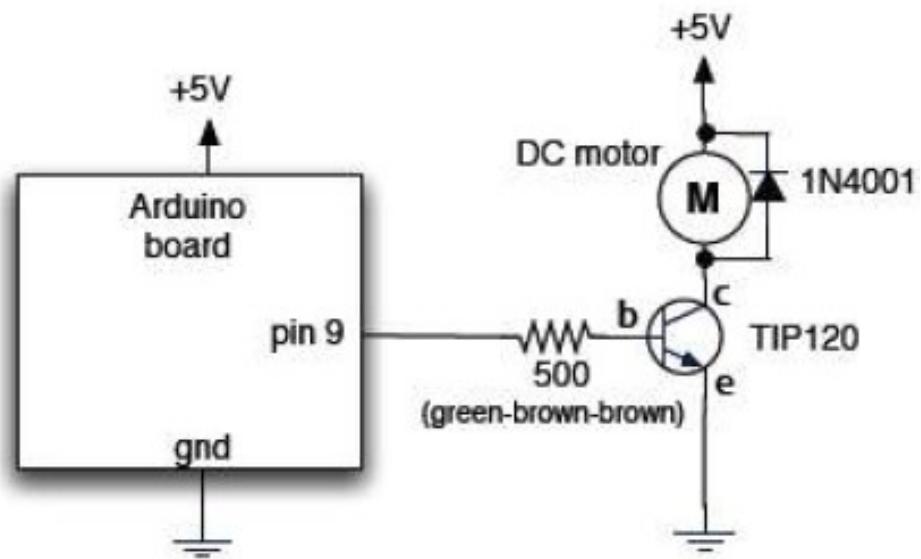
<http://www.marulaberry.co.za/index.php/tutorials/code/pulse-width-modulation/>

On utilise un transistor pour piloter le moteur.

On envoie un signal dont le rapport cyclique varie en fonction de la vitesse désirée.

La diode dite « de roue libre » permet d'évacuer le courant créé lorsque le moteur ralentie alors qu'il n'est plus alimenté.

La tension d'alimentation du moteur peut être différente (bien entendu)



```
/*
 * Contrôle d'un moteur DC - en utilisant une sortie PWM.
 * Attention, les pin PWM 5 et 6 ont une limitation.
 */
int pinMoteur = 9; // pin contrôlant le moteur (PWM pin)

void setup(){
  //En PWM il ne faut pas assigner la pin en output
}
void loop(){
  MoteurVitessePWM();
  MoteurAccelerationPWM();
}

// Contrôle de la vitesse du moteur en contrôlant le % du cycle de service PWM
// sur la pin 9 (de 0 à 255 pour le % du cycle de service).
//
void MoteurVitessePWM(){
  int vitesse1 = int(255) / 3; // High 33% du cycle
  int vitesse2 = int(255) / 2; // High 50% du cycle
  int vitesse3 = 2 * 255 / 3;
  int vitesse4 = (int)255; // High 100% du cycle
  analogWrite( pinMoteur, vitesse1 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse2 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse3 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse4 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse3 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse2 ); delay( 1000 );
  analogWrite( pinMoteur, vitesse1 ); delay( 1000 );
  analogWrite( pinMoteur, LOW );
}

// Contrôle plus fin de l'accélération du moteur via PWM.
// NB: Selon la qualité du moteur, celui-ci peut avoir du mal à décoller
// lorsque le % du cycle de service est assez bas.
// Un option est d'envoyer une impulsion pour démarrer/décoller le moteur.
void MoteurAccelerationPWM(){
  // Impulsion de démarrage (75%)
  // analogWrite( pinMoteur, 191 );
  //delay(50); // Acceleration
  for( int i = 30; i<= 255; i++ ){
    analogWrite( pinMoteur, i );
    delay(50); // delay pour avoir un progression
  }
  // pause de 2 secondes a plein régime
  delay( 2000 );
  // Deceleration
  for( int i = 255; i>=0; i-- ){
    analogWrite( pinMoteur, i );
    delay(50); // delay pour avoir un progression
  }
}
```

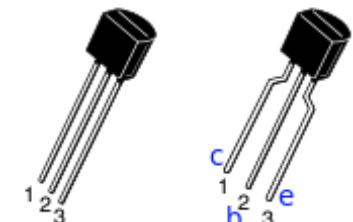
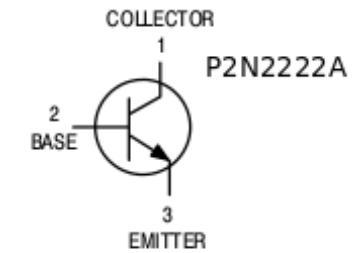
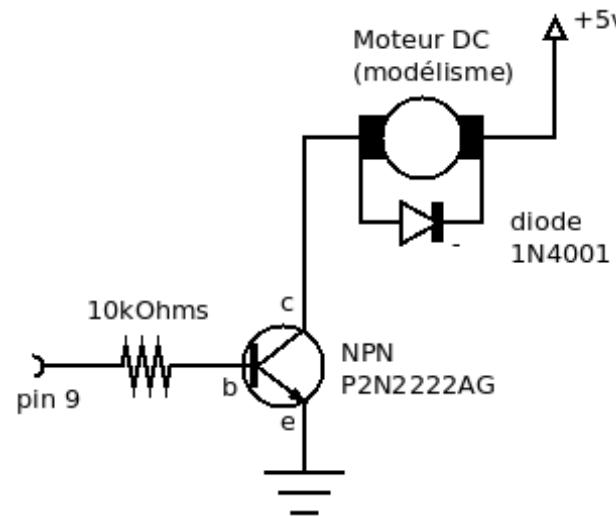
<http://arduino103.blogspot.com/2011/05/controle-moteur-dc-via-transistor-pwm.html>

## Contrôle Moteur DC via transistor (PWM):

Comme nous l'avons vu, le contrôle de la vitesse en analogique n'est pas vraiment approprié car il faut en effet vaincre l'inertie au démarrage.

Avec le MÊME MONTAGE, il est possible d'affiner le contrôle de la vitesse avec le PWM:

### MONTAGE:



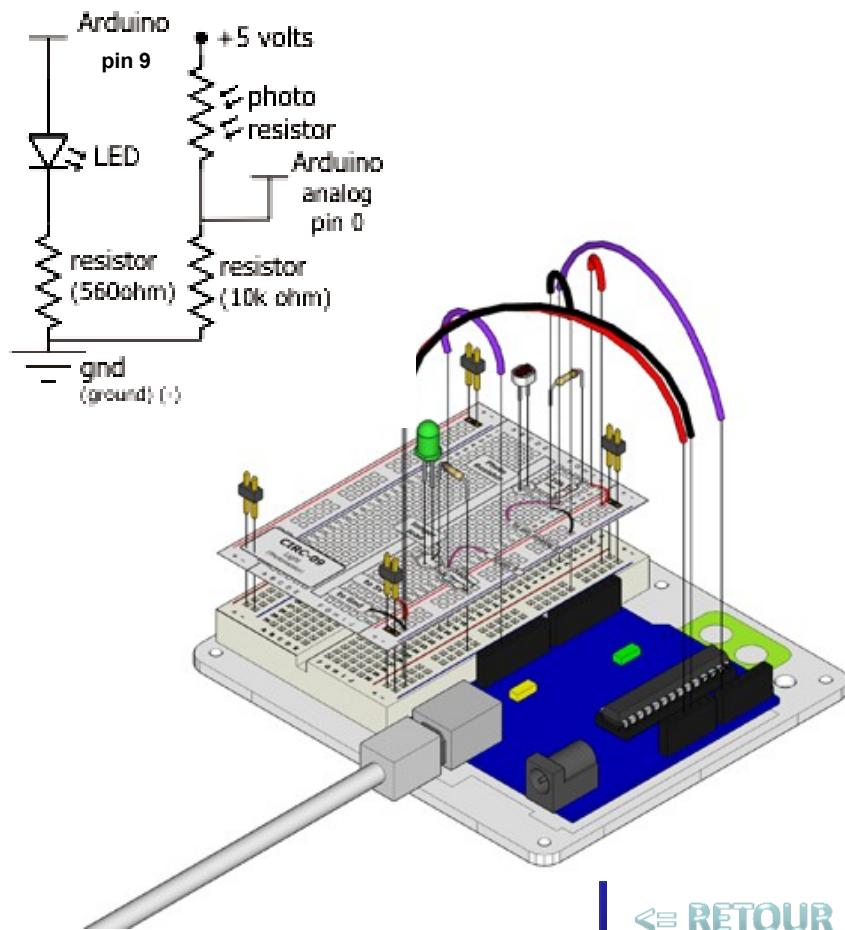
Voir également les notes complémentaires disponibles dans l'article "[Contrôle Moteur DC via transistor \(digital et analogique\)](#)», (<http://arduino103.blogspot.com/2011/05/controle-moteur-dc-via-transistor.html>)

Si l'homme peut contrôler un potentiomètre, comment faire pour que ce soit l'environnement qui contrôle (e.g. la lumière). Nous prendrons une photo-résistance (sa résistance est fonction de la lumière reçue). Or l'Arduino ne peut pas directement mesurer une résistance (il ne détecte qu'une tension) nous avons donc mis en place un diviseur de tension. La mesure se fait sur une entrée analogique.

**La résistance de la photorésistance diminue lorsque la lumière augmente**

```
/* A simple program that will change the intensity of a LED based on the
* amount of light incident on the photo resistor. */
//PhotoResistor Pin
int lightPin = 0; //the analog pin the photoresistor is connected to the
//photoresistor is not calibrated to any units so this is simply a raw sensor value
//(relative light)
int ledPin = 9; //the pin the LED is connected to
//we are controlling brightness so, we use one of the PWM (pulse
//width modulation pins)
void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT); //sets the led pin to output
}

void loop()
{
  int lightLevel = analogRead(lightPin); //Read the lightlevel
  Serial.print(lightLevel); //Affiche le lightlevel sur le moniteur série
  //ajuster les valeurs en fonction du lightlevel: adjust 180 to 610 to span 0 to 255
  lightLevel = map(lightLevel, 250, 700, 0, 255);
  Serial.print(","); Serial.println(lightLevel);
  delay(50);
  lightLevel = constrain(lightLevel, 0, 255); //make sure the value is between 0 et 255
  analogWrite(ledPin, lightLevel); //write the value
}
```



<http://www.oomlout.com/a/products/ardx/circ-09>

← RETOUR

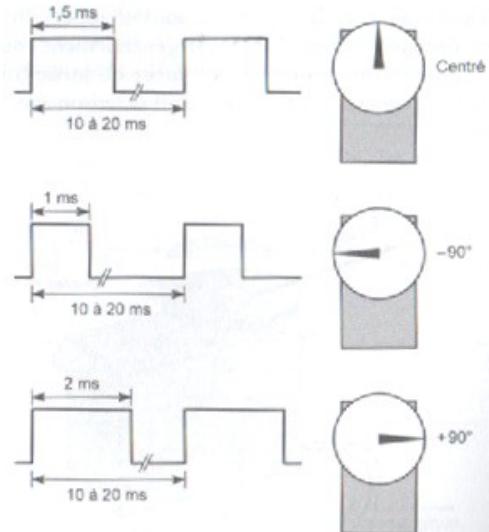
**Un servo-moteur est un type de moteur électrique.**

C'est un dispositif utilisé en modélisme pour contrôler la direction d'une voiture télécommandée.

Sur un servo-moteur, l'angle peut varier d'un angle fixe entre 0 et 180° en fonction du signal envoyé.

**Un servo-moteur comprend :**

- Un moteur électrique (continu), généralement assez petit.
- Des engrenages réducteur en sortie du ce moteur (pour avoir moins de vitesse et plus de couple ou de force).
- Un capteur type "potentiomètre" raccordé sur la sortie (résistance qui varie en fonction de l'angle, ce qui permet de mesurer l'angle de rotation sur l'axe de sortie).
- Un asservissement électronique pour contrôler la position/rotation, de cet axe de sortie pour le maintenir à la bonne position.



### Commander un servo-moteur ?

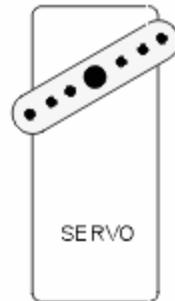
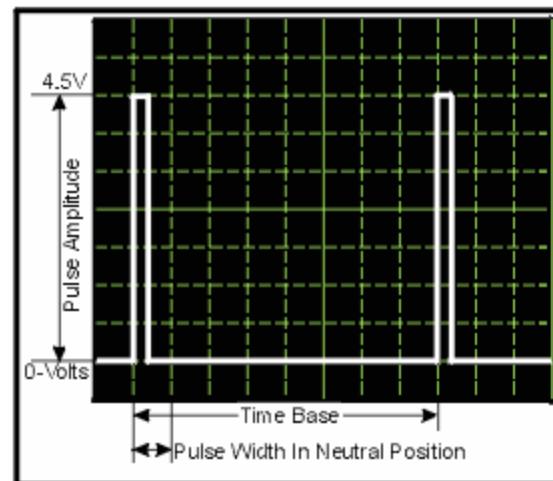
Il suffit d'envoyer une impulsion dont la durée déterminera l'angle du servo-moteur.

Ce temps d'impulsion est de quelques ms doit être répété à intervalle régulier (toutes les 10 ms à 20ms) pour conserver la position.

### Valeurs standards:

**1 ms = -90° | 1.5 ms = 0° (position centré) | 2 ms = +90°**

Le graphique animé ci-dessous montre la correspondance entre la longueur d'impulsion et l'angle du servo-moteur.



# CABLER UN SERVOMOTEUR

Un servo-moteur se raccorde avec seulement 3 fils:  
la masse, le +5V et la commande par impulsion.

Le fil NOIR est connecté au 0V ou GND.

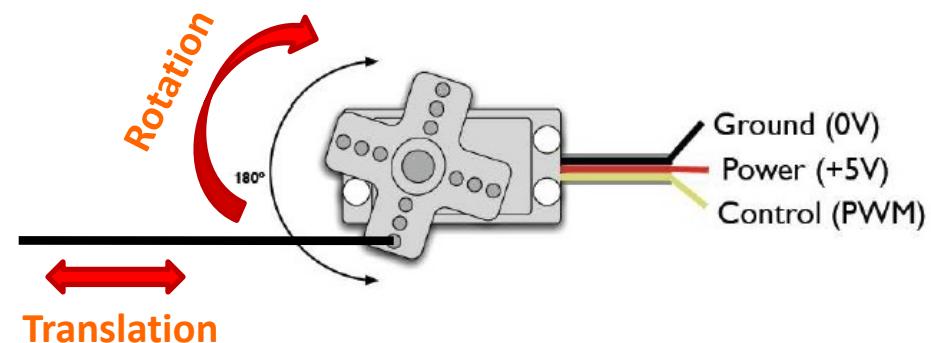
Le fil ROUGE est connecté au 5V.

Le fil JAUNE ou BLANC est connecté à une sortie numérique  
PWM

On envoie un signal que l'on fait varier en fonction du sens et  
de la position désirée.

On câble le servomoteur sur une des sorties numériques  
PWM: D11, D10, D9, D6, D5.

Programmer soit même la commande  
d'impulsion est toujours possible mais cela  
serait fastidieux,  
Arduino dispose de la librairie Servo.h qui  
permet de prendre facilement le contrôle.



Ce sous programme « servo.h » doit être inclus dans le programme, il est alors facile de le commander.

//File > Examples > Library-Servo > Sweep (example from the great arduino.cc site  
//check it out for other ideas)

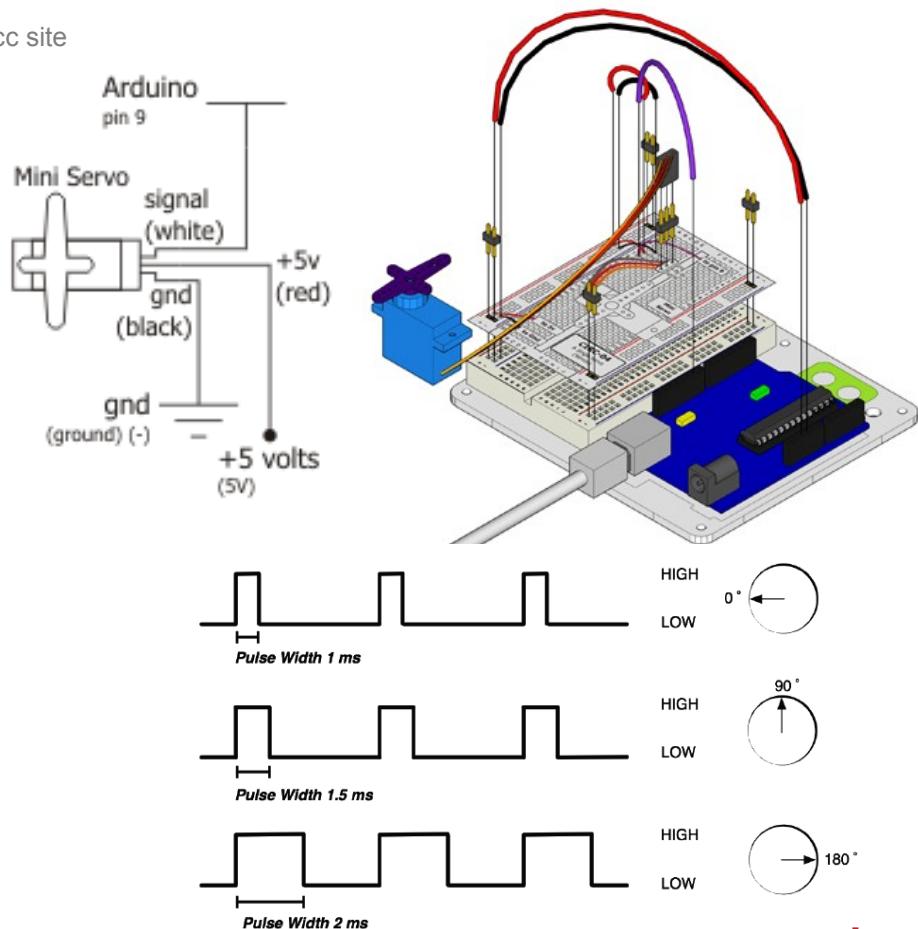
```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
// a maximum of eight servo objects can be created

int pos = 0; // variable to store the servo position

void setup()
{
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}

void loop()
{
  for(pos = 0; pos < 180; pos += 1) // goes from 0 degrees to 180 degrees
  {
    // in steps of 1 degree
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }

  for(pos = 180; pos>=1; pos-=1) // goes from 180 degrees to 0 degrees
  {
    myservo.write(pos); // tell servo to go to position in variable 'pos'
    delay(15); // waits 15ms for the servo to reach the position
  }
}
```



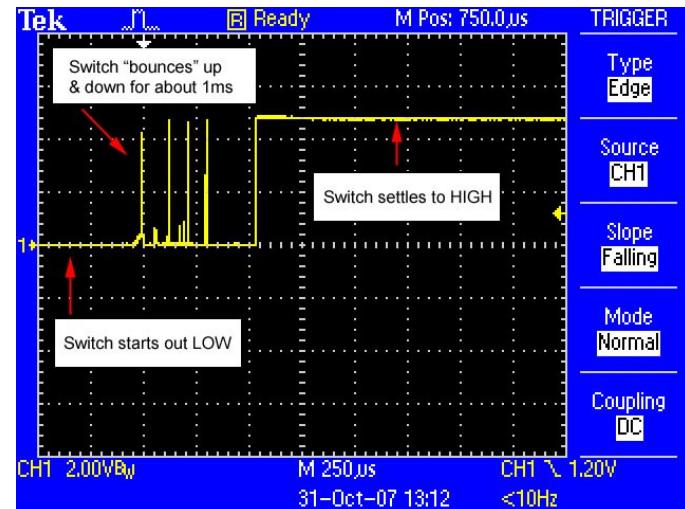
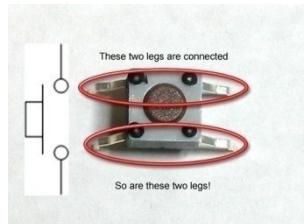
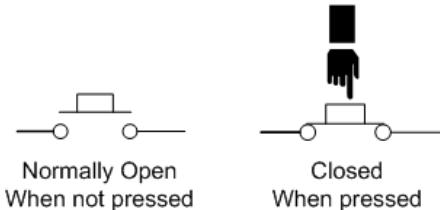
**Attach():** permet de définir et attacher la connexion d'un servo  
**Detach():** évidemment le contraire  
**Write():** fonction principale de commande du servo, elle permet de générer le train d'impulsion, le paramètre est la la position angulaire de 0 à 180°(pas la largeur d'impulsion) 90° est la position de repos ou médiane  
**Read():** permet de connaitre la position du servo en renvoyant la valeur du dernier appel au Write()

## III-5 DEPARASITAGE OU DEBOUNCING

# DEBOUNCING: DÉPARASITAGE DES BOUTONS

Lorsque l'on presse/relâche un bouton, il y a souvent l'apparition de parasites pendant quelques ms.

Ce parasites n'apparaissent que durant les moments où l'on enfonce/relâche le bouton poussoir.



Si l'on compte le nombre de pressions (pour faire un compteur), ces parasites viennent justement perturber le bon fonctionnement du logiciel. Le problème est matériel... et les parasites viennent ajouter des pressions « fantômes ».

Il y a plusieurs façons de corriger le problème:

**LOGICIELLE (Software debouncing)** => Introduire un délai logiciel dans le programme

**MATERIELLE (Hardware debouncing)** => Utiliser une capacité de déparasitage (Hardware debouncing) (et éventuellement un trigger de Schmidt Inverseur (Jeremy Blum Tutorial))

Voir l'article d'IkaLogic.com expliquant comment déparasiter des circuits (façon logiciel et matérielle)

Ou l'article de LadyADA <http://www.ladyada.net/learn/arduino/lesson5.html>

# SOFTWARE DEBOUNCING OU DÉPARASITAGE LOGICIEL

Le phénomène transitoire ne durant que quelques ms, il suffirait de faire deux lectures successives de l'entrée (après un délai de quelques ms) et de s'assurer qu'il ne s'agit pas d'une phase transitoire. La lecture de l'état de l'entrée 10ms plus tard serait donc identique à celle 10 ms plus tôt: **Personne n'arrivant à presser et relâcher un bouton en moins de 10 ms !!!**

*/\* Debounce:* each time the input pin goes from LOW to HIGH (e.g. because of a push-button press), the output pin is toggled from LOW to HIGH or HIGH to LOW. There's a minimum delay between toggles to debounce the circuit (i.e. to ignore noise).

The circuit: \* LED attached from pin 13 to ground | \* pushbutton attached from pin 2 to +5V | \* 10K resistor attached from pin 2 to ground

\* Note: On most Arduino boards, there is already an LED on the board connected to pin 13, so you don't need any extra components for this example.

Created 21 November 2006, by David A. Mellis, modified 3 Jul 2009, by Limor Fried and J. Grisolia et B. Detroussel Oct 2011 \*/

// constants won't change. They're used here to set pin numbers:

const int buttonPin = 2; // the number of the pushbutton pin

const int ledPin = 13; // the number of the LED pin

// Variables will change:

int ledState = HIGH; // the current state of the output pin

int buttonState; // the current reading from the input pin

int lastButtonState = LOW; // the previous reading from the input pin

int compteur\_filtre; //compteur filtré

int compteur\_non\_filtre; //compteur non filtré

// the following variables are long's because the time, measured in miliseconds, will quickly become a bigger number than can be stored in an int.

long lastDebounceTime = 0; // the last time the output pin was toggled

long debounceDelay = 50; // the debounce time; increase if the output flickers

**void setup()** {

Serial.begin (9600);

pinMode(buttonPin, INPUT);

pinMode(ledPin, OUTPUT);

}

**void loop()** {

// read the state of the switch into a local variable:

int reading = digitalRead(buttonPin);

// check to see if you just pressed the button (i.e. the input went from LOW to HIGH),

// and you've waited long enough since the last press to ignore any noise: If the switch changed, due to noise or press

if (reading != lastButtonState) {

lastDebounceTime = millis(); // reset the debouncing timer

if (reading == 1) { compteur\_non\_filtre = compteur\_non\_filtre + 1; }

}

if ((millis() - lastDebounceTime) > debounceDelay)

{ // whatever the reading is at, it's been there for longer than the debounce delay, so take it as the actual current state:

if ((reading == 1) && (reading != buttonState)) { compteur\_filtre = compteur\_filtre + 1; }

buttonState = reading;

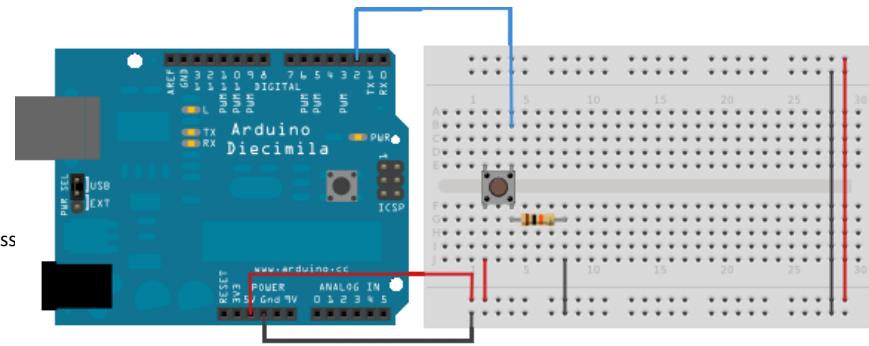
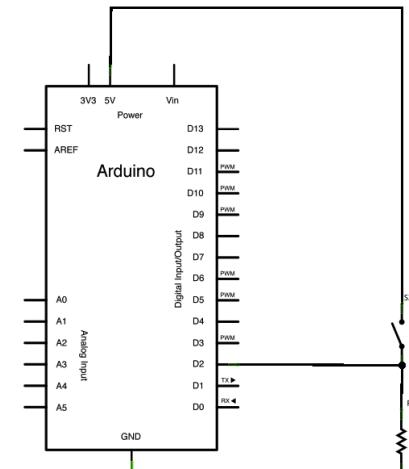
}

digitalWrite(ledPin, buttonState); // set the LED using the state of the button

lastButtonState = reading; // save the reading. Next time through the loop, it'll be the lastButtonState

Serial.print("Compteurs:"); Serial.print(compteur\_non\_filtre); Serial.print(", "); Serial.println(compteur\_filtre);

}



<http://www.arduino.cc/en/Tutorial/Debounce>

Le déparasitage se fait à l'aide d'une capacité qui absorbera les impulsions parasites.

Mais à elle seule, elle n'est pas suffisante.

Dans le cas d'un montage pull-up, on place une capacité de  $100\text{nF}$  (en série avec une résistance de  $10\text{k}\Omega$ ) soit une constante de temps  $\tau=RC=1\text{ms}$

### Lorsque l'on presse le bouton:

La capacité est instantanément déchargée (court-circuité par le bouton) et l'entrée passe en LOW.

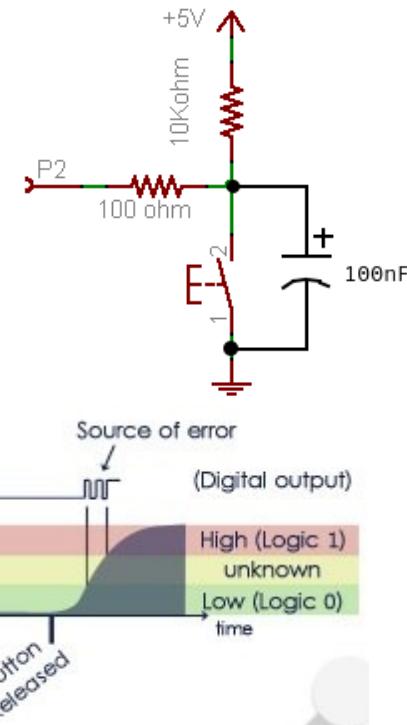
Si il y a des parasites (pics à +5 volts) au niveau du bouton, ces derniers seront absorbés par la capacité.

### Lorsque l'on relâche le bouton:

La tension est appliquée sur la capacité et cette dernière va se charger assez vite.

Cependant, la charge n'est pas instantanée mais progressive (dépend de  $\tau$ ) => les 5 volts sont appliqués progressivement à l'entrée =>

La courbe de charge passe un certain temps dans la zone d'incertitude où le micro-contrôleur ne sait pas si c'est toujours un 0 ou déjà un 1 logique. Le parasitage existe donc toujours au moment où l'on relâche le bouton (mais sous une autre forme).



Pour éviter ce nouveau type de parasitage, il faut insérer un trigger de Schmitt sur l'entrée.

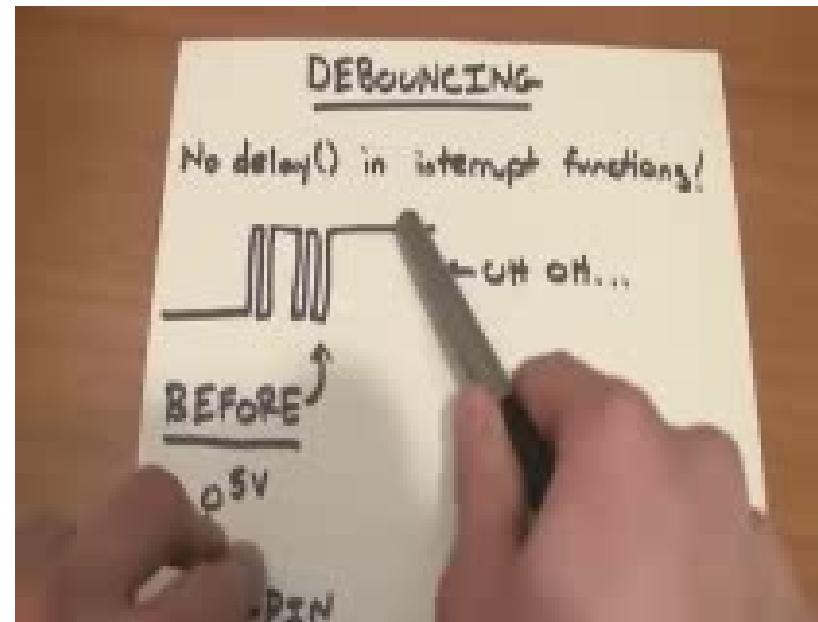
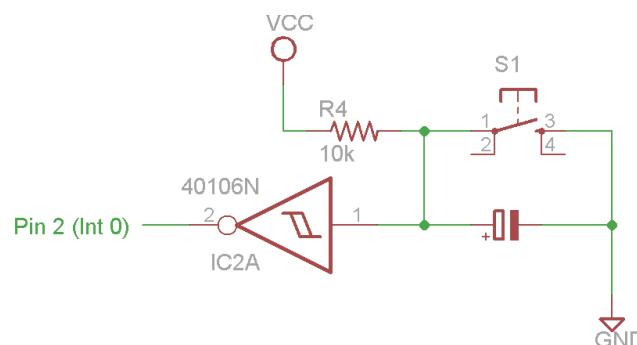
Ainsi, tant que la tension sera en dessus du seuil minimal du 1 logique, le trigger ne change pas d'état vers 1... et inversement tant que la tension ne sera pas passée sous le seuil minimum du trigger (0 logique), le trigger ne repassera pas à 0 (cf transparent 52).

# HARDWARE DEBOUNCING

//Program by Jeremy Blum [www.jeremyblum.com](http://www.jeremyblum.com)  
 //Debounced Switch Input via hardware

```
int switchPin = 2;
int ledPin = 13;
boolean lastButton = LOW;
boolean ledOn = false;
void setup()
{
  pinMode(switchPin, INPUT);
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
void loop()
{
  Serial.println(digitalRead(switchPin));
  if (digitalRead(switchPin) == HIGH && lastButton == LOW)
  {
    ledOn = !ledOn;
    lastButton = HIGH;
  }
  else
  {
    //lastButton = LOW;
    lastButton = digitalRead(switchPin);
  }

  digitalWrite(ledPin, ledOn);
}
```



## PART LIST:

- Inverting Schmitt Trigger: <http://us.element-14.com/stmicroelectronics/m74hc14b1r/ic-hex-inverter-schmitt-trigger/dp/89K0862>
- 10k ohm Resistor: <http://us.element-14.com/multicomp/mccfr0w4j0103a50/resistor-carbon-film-10kohm-250mw/dp/58K5002>
- 10uF Capacitor: <http://us.element-14.com/multicomp/mcgpr100v106m6-3x11/capacitor-alum-elect-10uf-100v/dp/70K9661>

## III – 6 - 1 LES INTERRUPTIONS

-**DEFINITION:** une interruption, comme son nom l'indique, interrompt un programme en cours (programme principal) pour exécuter un sous-programme au microcontrôleur grâce à l'interrupt service routine (ISR).

- Celui-ci se termine par une instruction de retour d'interruption qui permet au microcontrôleur de reprendre le programme principal où il l'avait quitté.

Une interruption peut survenir de deux manières :

- 1 - Interruption matérielle: elle est sollicitée par le matériel pour effectuer un traitement (lire une touche du clavier par exemple). Dans ce cas, il déclenche une IRQ (Interrupt ReQuest) possédant un numéro propre au port auquel le matériel est connecté.
- 2 - Interruption logicielle: elle est sollicitée par le logiciel en cours d'exécution.

**Pourquoi faire ?** Les interruptions sont un moyen très performant pour surveiller un évènement extérieur. L'utilisation des interruptions libère notamment le microcontrôleur pour faire d'autres choses tant qu'un évènement attendu ne survient pas.

**Exemple:** Soit un programme dans lequel un bouton poussoir doit provoquer un changement:

**En programmation classique:** on met un test du bouton dans le programme, sans être certain qu'on ne manquera pas l'appui par exemple très bref.

**En programmation interruption:** il n'y a plus besoin de test, car l'appui du bouton générera l'appel au sous-programme

**Autre exemple:** d'autres capteurs ont un rôle d'interface dynamique semblable, tel qu'un capteur de sons essayant détecter un bruit ou un capteur infra-rouge (photo-transistor) essayant de détecter un obstacle

Pour aller plus loin: [http://www.iutc3.unicaen.fr/~fougep/assembleur/les\\_chapitres\\_du\\_cours/interruptions.html](http://www.iutc3.unicaen.fr/~fougep/assembleur/les_chapitres_du_cours/interruptions.html)  
[http://www.technologuepro.com/microprocesseur/chap5\\_microprocesseur.htm](http://www.technologuepro.com/microprocesseur/chap5_microprocesseur.htm)

## III – 6 - 1 LES INTERRUPTIONS MATERIELLES

**attachInterrupt (interruption, fonction, mode):** spécifie la fonction à appeler lorsqu'une interruption externe survient.

Les plupart des Arduino ont 2 interruptions externes : n°0 sur Pin numérique 2 et n°1 sur Pin numérique 3  
Sauf, l'Arduino Mega qui en a quatre de plus : n°2, 3, 4, 5 sur Pin 21, 20, 19, 18

## Paramètres:

**interruption :** le numéro de l'interruption (type int)

**fonction:** la fonction à appeler quand l'interruption survient; la fonction doit recevoir aucun paramètre et ne renvoie rien. Cette fonction est également appelée une routine de service d'interruption (ou ISR).

**mode :** définit la façon dont l'interruption externe doit être prise en compte. Quatre constantes ont des valeurs prédéfinies valables :

**LOW** : pour déclenchement de l'interruption lorsque la broche est au niveau BAS

**CHANGE** : pour déclenchement de l'interruption lorsque la broche change d'état BAS/HAUT

**RISING** : pour déclenchement de l'interruption lorsque la broche passe de l'état BAS vers HAUT (front montant)

**FALLING** : pour déclenchement de l'interruption lorsque la broche passe de l'état HAUT vers l'état BAS (front descendant)

**Exemple:** attachInterrupt(0, blink, CHANGE); //attache l'interruption externe n°0 (Pin2) à une fonction blink

## Attention:

- A l'intérieur de la fonction attachée à l'interruption, la fonction **delay** ne fonctionne pas et la valeur renvoyée par **millis** ne s'incrémente pas. **POURQUOI ?**

- Les données séries reçues pendant l'exécution de la fonction sont perdues.

- Il faut déclarer « volatile » toutes les variables utilisées dans la fonction attachée à l'interruption: permet de rendre la variable accessible depuis tous contextes, y compris dans des interruptions.

# DERRIERE LE RIDEAU...

## 12.4 Interrupt Vectors in ATmega328 and ATmega328P Vecteurs appelés par l'interruption:

Table 12-6. Reset and Interrupt Vectors in ATmega328 and ATmega328P

VectorNo.	Program Address <sup>(1)</sup>	Source	Interrupt Definition
1	0x0000 <sup>(1)</sup>	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	PCINT0	Pin Change Interrupt Request 0
5	0x0008	PCINT1	Pin Change Interrupt Request 1
6	0x000A	PCINT2	Pin Change Interrupt Request 2
7	0x000C	WDT	Watchdog Time-out Interrupt
8	0x000E	TIMER2 COMPA	Timer/Counter2 Compare Match A

### 13.2.1 EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	EICRA
(0x69)	—	—	—	—	ISC11	ISC10	ISC01	ISC00	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 13-1. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

### 13.2.2 EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	EIMSK
0x1D (0x3D)	—	—	—	—	—	—	INT1	INT0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

### 13.2.3 EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	EIFR
0x1C (0x3C)	—	—	—	—	—	—	INTF1	INTF0	
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

## External interrupt

```
// on a initialisé un poussoir et une led
//Setup - INT1
EICRA = 1<<ISC11 ; // falling edge
EIMSK = 1<<INT1 ;
sei() ;
```

ISC11 à 1 et ISC10 à 0

Initialiser les registres

```
//
ISR(INT1_vect) {
    LedToggle ;
}
```

Dire au microprocesseur de partir dans la bonne adresse (vecteur)

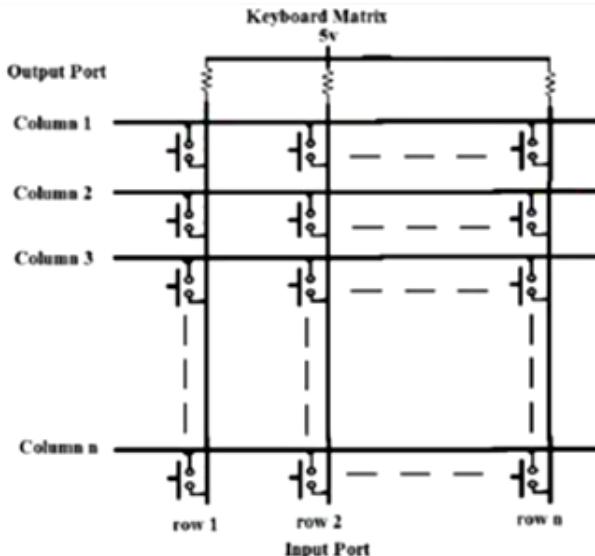
## Arduino attachInterrupt

```
void setup() {
    pinMode ...
    attachInterrupt (pin/no, FaireQqch, LOW);
}
```

```
void FaireQqch () {
    LedToggle ;
}
```

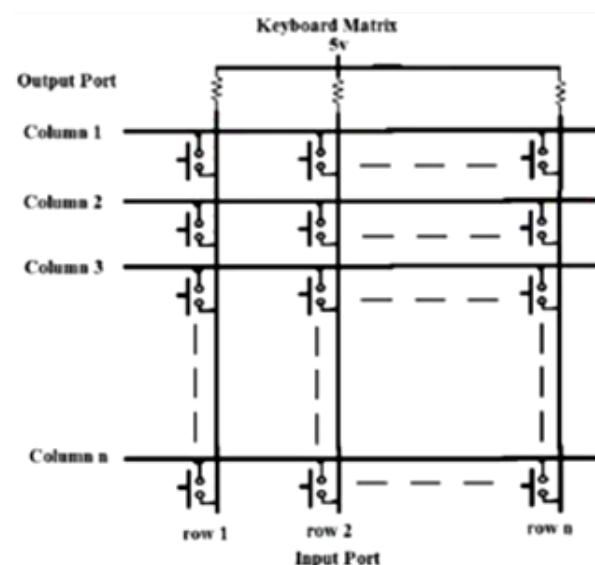
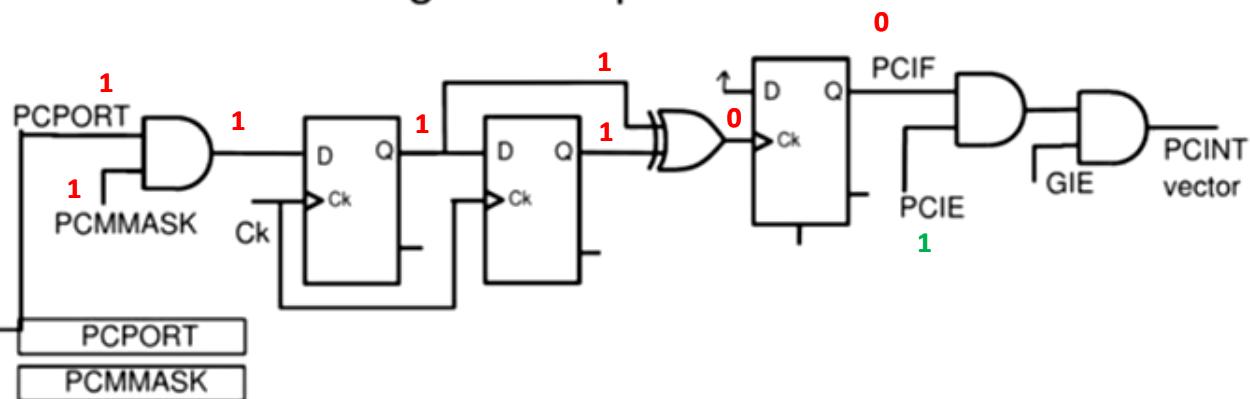
Version Arduino plus simple, mais bien entendu plus lente car ce sont alors des procédures qui testent...

# POUR ALLER PLUS LOIN...



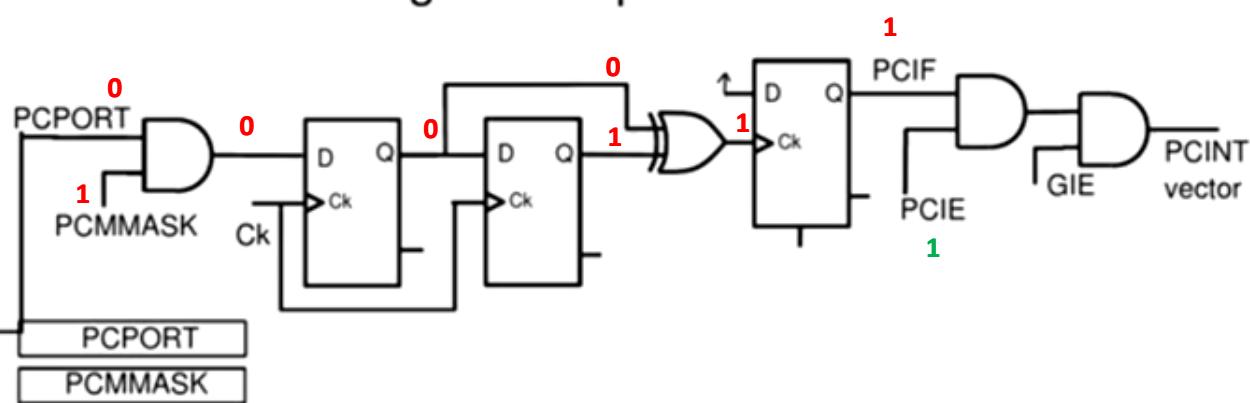
Avec pull-up: niveau logique 1 sans appui

## Pin change interrupt



Appui: 1 => 0

## Pin change interrupt

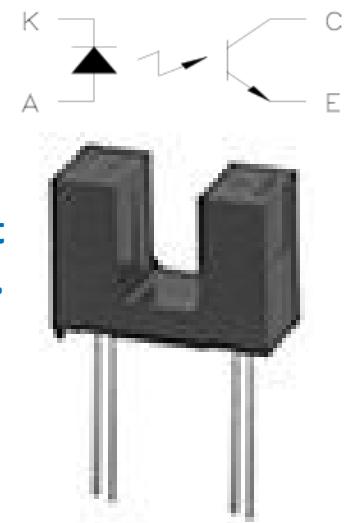


# PROGRAMME INTERRUPTIONS

Un opto-coupleur (ou photocoupleur) en fourche associe dans un même boîtier plastique:

- 1 - une LED qui émet de la lumière infra-rouge et
- 2 - un photo-transistor qui détecte la lumière émise par la LED.

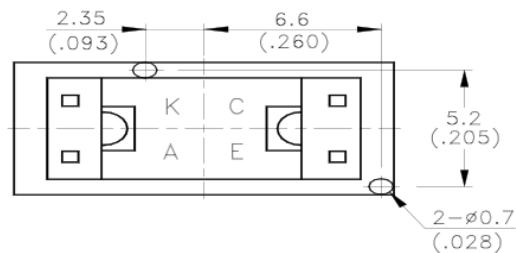
Les deux composants sont faces à faces et séparés par une fente dans laquelle il est possible de faire passer une roue encodeuse ou un objet rotatif quelconque. L'ensemble forme une barrière photo-électrique miniature.



Ce composant dispose de 4 broches :

- 2 premières broches notées A et K correspondent aux broches de la LED
- 2 autres broches notées C et E correspondant au collecteur et à l'émetteur du phototransistor. La base du transistor est ici une "photobase" qui au lieu de recevoir un courant reçoit les photons en provenance de la LED

*Exemple optocoupleur en fourche: Liton LTH 307-01*

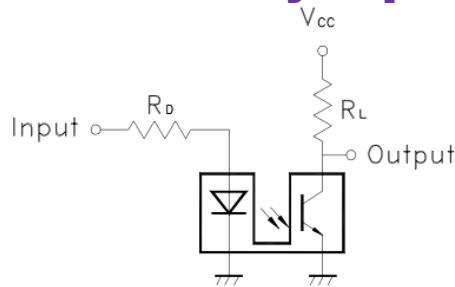


En fixant astucieusement la valeur des résistances utilisées, on obtiendra avec  $V_{CC}=5V$  :

- 0V (ou niveau BAS) en sortie du phototransistor lorsqu'il sera éclairé
- 5V (ou niveau HAUT) en sortie du phototransistor lorsqu'il ne sera pas éclairé.

=> il sera ainsi possible de détecter le passage d'un objet dans la fente.

Comment choisir  $R_D$  et  $R_L$  ?



Pour la résistance  $R_D$  de la LED :

la chute de tension aux bornes de la LED est ~1,1 V (cf datasheet)

=> pour avoir une  $I_{LED} = 15mA$  (sous  $V_{CC}=5V$ ):  $R_D = U/I = (5-1,1)/0.015 = 260 \Omega$

=>  $270\Omega$  valeur standardisée

Pour la résistance  $R_L$  du photo-transistor, l'objectif est :

1 - 0V sur  $R_L$  (i.e. Output à 5V) quand le photo-transistor n'est pas éclairée et

2 - ~5V sur  $R_L$  (i.e. Output à 0V, plus précisément ~1V) lorsqu'il sera éclairé.

- Si  $I_{LED} = 0mA$ :

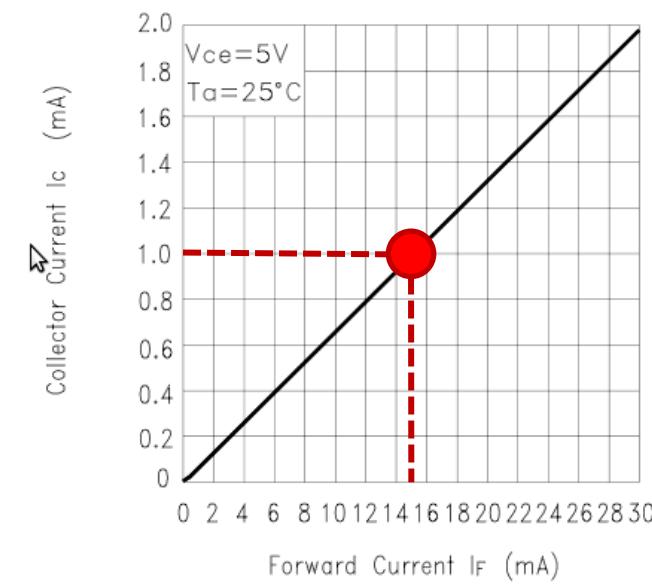
$R_L$  n'a pas d'importance quand le photo-transistor n'est pas éclairé car  $I_C = 0mA$  et donc  $U_L = R_L \times I_C = R_L \times 0 = 0V$

- Si  $I_{LED} = 15mA$ : =>  $I_C = 1mA$  et  $U_{CE} \sim 1V$  (typique d'une jonction PN).

Pour 5V sur  $R_L$  avec  $I_R = 1mA$  vaut :  $R_L = (5V-1V)/0.001 = 4 k\Omega \Rightarrow 4,7 k\Omega$  valeur standardisée.

On prendra donc :  $R_D = 270 \Omega$  et  $R_L = 4.7 k\Omega$

Fig.3 Collector Current vs. Forward Voltage



# PROGRAMME INTERRUPTIONS

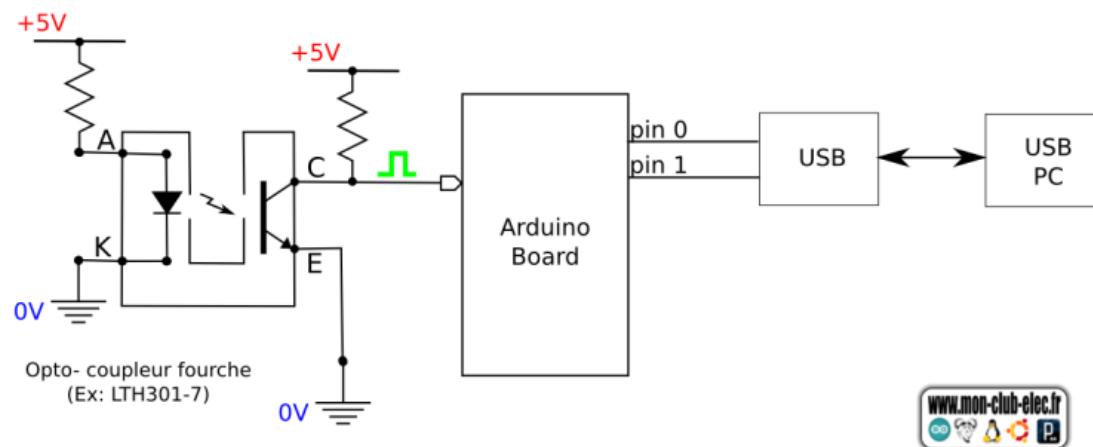
Réalisez le programme suivant:

Le programme compte les impulsions en provenance d'un opto-coupleur en fourche en utilisant l'interruption externe n°0 sur la broche 2 de la carte Arduino.

Le résultat est affiché dans le Terminal Série du logiciel Arduino.

Ce programme utilise les fonctionnalités suivantes :

- Utilise la connexion série vers le PC (broches 0 et 1 (via le câble USB))
- Utilise l'interruption externe 0 (broche 2)



La routine de gestion de l'interruption doit être appelée à chaque fois qu'un front montant sera détecté sur la broche :

- La variable de comptage est alors incrémentée
- Le résultat est affiché dans la fenêtre Terminal du logiciel Arduino.

# PROGRAMME INTERRUPTIONS

```
// Programme interruption basé sur le site www.mon-club-elec.fr Auteur du Programme : X.
HINAULT - Tous droits réservés Programme écrit le : 13/2/2011.
// ----- Que fait ce programme ?
/* Ce programme compte les impulsions en provenance d'un opto-coupleur en fourche
en utilisant l'interruption externe n°0 sur la broche 2 de la carte Arduino.
Le résultat est affiché dans le Terminal Série du logiciel Arduino. */
// --- Fonctionnalités utilisées ---
// Utilise la connexion série vers le PC
// Utilise l'interruption externe 0 (broche 2)
// Utilise optocoupleur infra-rouge en fourche, type LTH301-7,
// --- Déclaration des variables globales ---
volatile int comptageImpulsion=0; // variable accessible dans la routine interruption externe 0

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

Serial.begin(115200); // initialise connexion série à 115200 bauds
attachInterrupt(0, gestionINT0, RISING); // attache l'interruption externe n°0 à la fonction
gestionINT0()
// mode déclenchement possibles = LOW, CHANGE, RISING, FALLING
} // fin de la fonction setup()
// ****
*****
```

```
void loop(){ // debut de la fonction loop()
// tout se passe dans la fonction de gestion de l'interruption externe
} // fin de la fonction loop()

// ----- fonction de gestion l'interruption externe n°0 (broche 2) -----
// cette fonction est appelée à chaque fois que l'interruption a lieu selon le mode configuré
//(LOW, //CHANGE, RISING, FALLING)
```

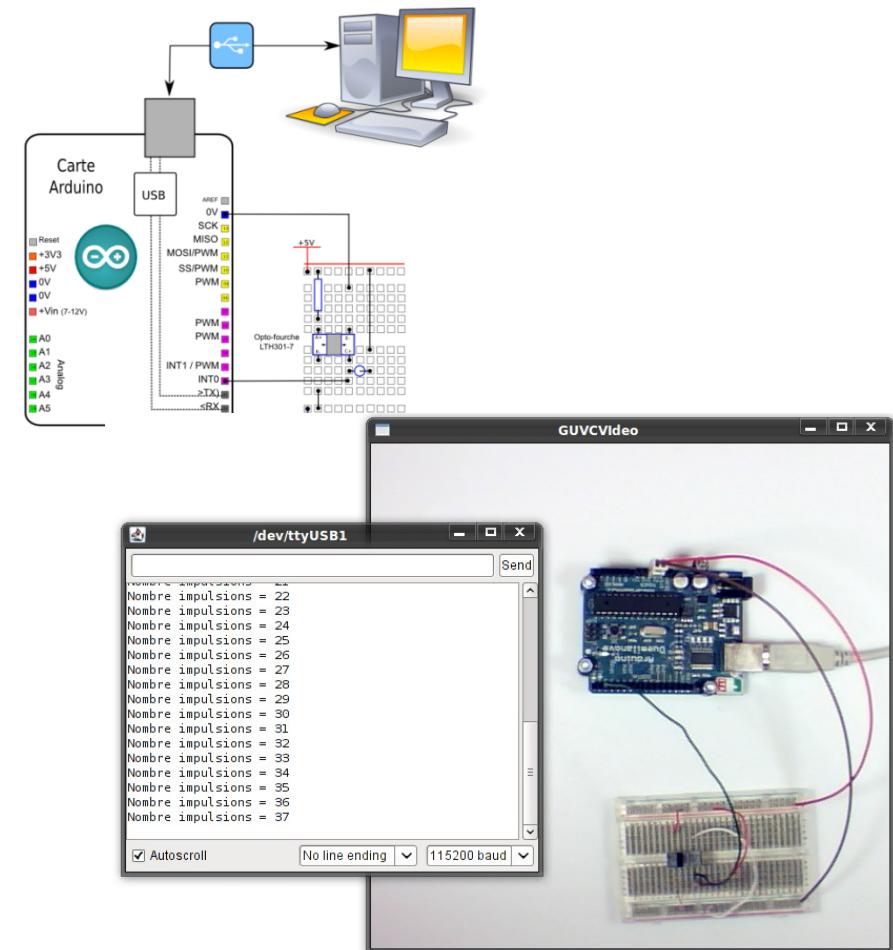
```
void gestionINT0(){ // la fonction appelée par l'interruption externe n°0
comptageImpulsion=comptageImpulsion+1; // Incrémente la variable de comptage
// ATTENTION : delay() et millis() non dispo ici - données série perdues
//--- affiche le nombre d'impulsions sur le port série
Serial.print("Nombre impulsions = ");
Serial.println(comptageImpulsion);
}
```

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheComptageInterruption](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheComptageInterruption)

Pour aller plus loin, interfaçage avec Processing: [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.OutilsProcessingProgGraphOscilloSimple](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.OutilsProcessingProgGraphOscilloSimple)

## ATTENTION:

REPLACEZ L'OPTOCOUPEUR PAR UN BOUTON POUSSOIR  
POUR METTRE EN EVIDENCE LES REBONDS !!!



RETOUR

## III – 6 - 1 LES INTERRUPTIONS LOGICIELLES

Comment réaliser des interruptions « logicielles » à intervalle régulier:  
**librairie MSTimer2**

Ce programme génère:

- une interruption toutes les secondes à l'aide du Timer 2 (8-bits) et
- allume brièvement une LED

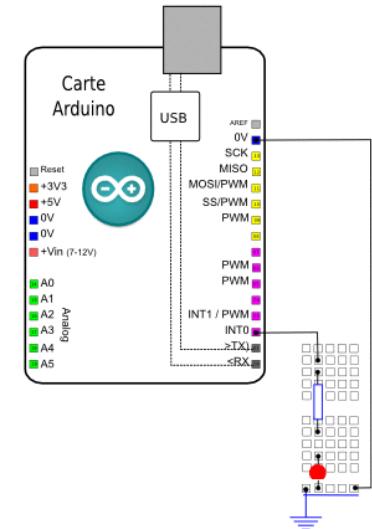
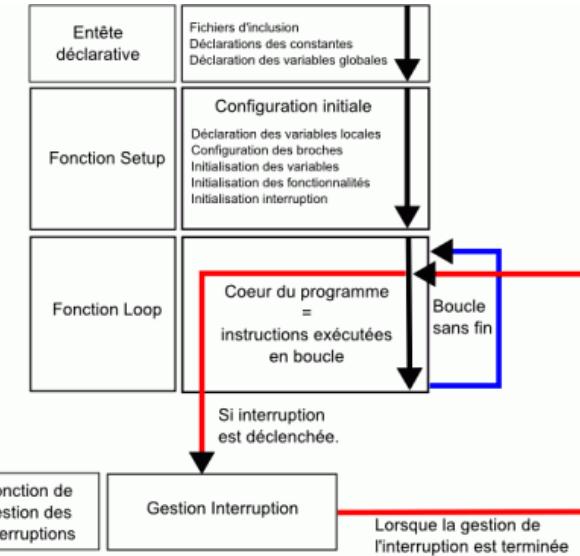
// --- Programme Arduino --- par X. HINAULT - Le 24/02/2010 www.mon-club-elec.fr

```
#include <MsTimer2.h> // inclusion de la librairie Timer2
const int LED=2; //declaration constante de broche

void setup() { // debut de la fonction setup()
pinMode(LED, OUTPUT); //met la broche en sortie
// initialisation interruption Timer 2
MsTimer2::set(1000, InterruptTimer2); // période 1000ms
MsTimer2::start(); // active Timer 2
} // fin de la fonction setup()

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

***** Autres Fonctions du programme *****
void InterruptTimer2() { // debut de la fonction d'interruption Timer2
digitalWrite(LED, HIGH);
delayMicroseconds(10000); // la fonction delayMicroseconds ne bloque pas les interruptions
digitalWrite(LED, LOW);
}
```



Exemple: <http://blog.blinkenlight.net/experiments/removing-flicker/heartbeat/>

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinolInitiationInterruptionsTemporisationTimer2UneSeconde](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinolInitiationInterruptionsTemporisationTimer2UneSeconde)

## III – 6 - 2 LES TIMERS

En tant que programmeur Arduino, vous avez utilisé des timers et des interruptions sans le savoir, car tout le hardware bas niveau est caché par l'IDE Arduino.

De nombreuses fonctions utilisent les timers: `delay()`, `millis()`, `analogWrite()` pour le PWM, `tone()`.

Même la bibliothèque Servo utilise les timers et les interruptions.

**QU'EST-CE QU'UN TIMER ?** Un timer, ou pour être plus précis un timer/counter, est un morceau de hardware de l'ATMega qui se comporte comme une horloge afin de mesurer des événements temporels, compter, répéter, comparer.

L'ATMega 328 de l'Arduino UNO possède trois timers, appelés `timer0`, `timer1` et `timer2` alors que L'ATmega1280 ou le ATMEGA2560 de la série Arduino Mega en possède 6 (2\*8bits et 4\*16bits) :

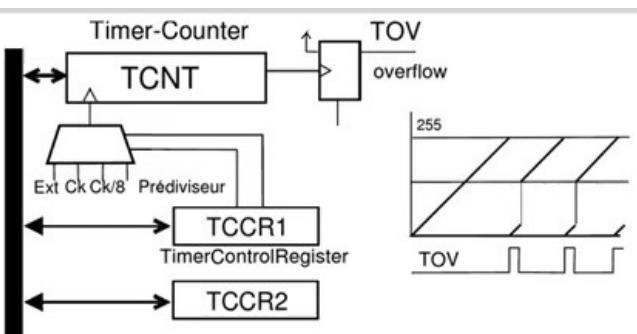
- Timer0: (8bit timer) utilisé par `delay()`, `millis()` et `micros()`. Si vous changez les registres de ce timer() cela pourrait influencer ces fonctions => attention
- Timer1: (16bit timer) utilisé par la librairie `Servo` sur l'Arduino Uno (timer5 sur l'Arduino Mega).
- Timer2: (8bit timer) comme le timer0. Utilisé par `tone()` dans l'Arduino Uno.
- Timer3, Timer4, Timer5: (16bit timers) seulement disponible sur l'Arduino Mega.

*Exemple: La fonction `tone()` utilise le timer2 => vous ne pouvez pas utiliser les PWM sur les pins 3 et 11*

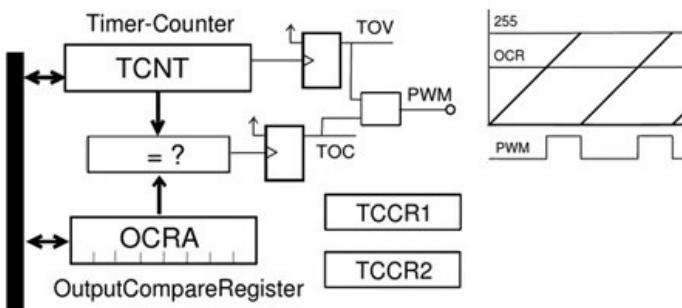
La grosse différence entre les timers 8 et 16 bits est bien entendu la résolution de l'horloge: 256 et 65536 valeurs respectivement pour le 8 et 16bits

Tous les timers dépendent de l'horloge système de votre Arduino (e.g. 16MHz pour le UNO) MAIS certains peuvent fonctionner à des fréquences différentes: exemple l'Arduino Pro à seulement 8MHz  
=> donc soyez prudent lorsque vous écrivez vos propres fonctions de timers.

Les timers sont programmés par des registres spéciaux, utilisent des prescalers et possède plusieurs modes de fonctionnement que nous allons voir maintenant:



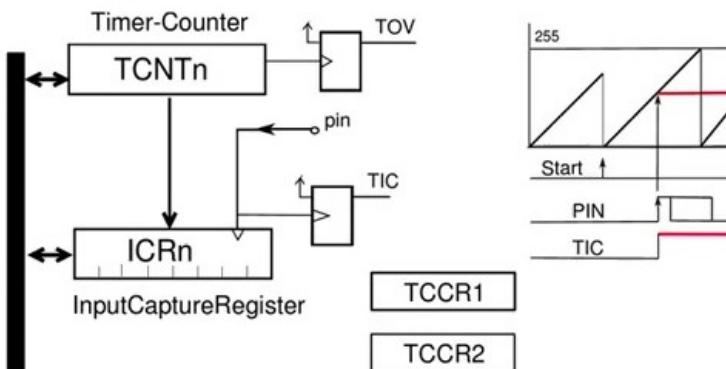
- 1 – Fréquence du timer-compteur définie par prescaler
  - 2 - Initialiser les registres,
  - 3 - Attendre l'activation de TOV (quand le compteur arrive à la valeur fournie)
  - 4 - Remettre à zéro TOV
  - 5 - Réinitialiser TCNT pour la durée voulue
- => Mesurer des durées



La comparaison avec une ou deux valeurs intermédiaire donne beaucoup de flexibilité

⇒ PWM

*Dans le bloc logique: on démarre à l'activation de TOC et on finit à l'activation de TOV*



L'activation de la pin mémorise l'état du compteur dans ICRn et active le flag TIC  
=> Interruption hardware

**Registres des TIMERS:** vous pouvez modifier le comportement des timers via les registres timers. Les registres timers les plus importants sont les suivants:

- **TCCRxy - Timer/Counter Control Register.**  
*où l'on peut définir les prescalers et d'autres choses...*

- **TCNTx - Timer/Counter Register.**

*La valeur de l'horloge réelle est stockée ici.*

- **OCRx - Output Compare Register**

- **ICRx - Input Capture Register (seulement pour les 16bit timers).**

- **TIMSKx - Timer/Counter Interrupt Mask Register.**  
*Pour activer/désactiver les interruptions d'horloge.*

- **TIFRx - Timer/Counter Interrupt Flag Register.**  
*Indique une interruption d'horloge en attente.*

**Remarques:**

- le suffixe x représente le numéro du timer (0...2)
- le suffixe y représente la lettre de la sortie (A, B, C):

Par exemple **TIMSK1** (masque d'interruption du timer1) ou **OCR2A** (registre A de l'OCR du timer2).

#### 16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

#### 16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

#### 18.11.3 TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
(0xB2)	–	–	–	TCNT2[7:0]	–	–	–	–	TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

#### 16.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	–	–	–	OCR1A[5:8]	–	–	–	–	OCR1AH
(0x88)	–	–	–	–	OCR1A[7:8]	–	–	–	OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

#### 16.11.7 ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	–	–	–	ICR1[15:8]	–	–	–	–	ICR1H
(0x86)	–	–	–	–	ICR1[7:8]	–	–	–	ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

#### 15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### 16.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

#### 18.11.7 TIFR2 – Timer/Counter2 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

**Ref: Atmel 8-bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash: pages 132, 134, 160...**

## Sélection de la fréquence du timer et du cycle d'horloge:

$$\frac{f_{clk\_I/O}}{N.256} \quad \frac{f_{clk\_I/O}}{N.65536}$$

Des sources d'horloge différentes peuvent être sélectionnées indépendamment pour chaque timer

Exemple: valeur de déclenchement du timer si l'on veut changer l'état d'une pin à une fréquence désirée de f=2Hz avec le timer1 (16-bits), un prescaler de 256 et une fréquence horloge système de 16MHz:

**Valeur compteur= (16MHz/(256\*65536))/2Hz=31250**

Si le résultat du timer/compteur par rapport à son maximum est

OK (31250 <65536 => OUI) => on peut utiliser le timer1

Avec le timer 8bits: 31250>256 => on ne pouvait pas

Si échec dans l'un des deux cas, solution: choisissez un prescaler plus grand (=> on ralentit l'horloge).

Table 16-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk <sub>I/O</sub> /1 (No prescaling)
0	1	0	clk <sub>I/O</sub> /8 (From prescaler)
0	1	1	clk <sub>I/O</sub> /64 (From prescaler)
1	0	0	clk <sub>I/O</sub> /256 (From prescaler)
1	0	1	clk <sub>I/O</sub> /1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

Table 16-4. Waveform Generation Mode Bit Description<sup>(1)</sup>

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX

Pour qu'une interruption en attente soit en mesure d'appeler l'ISR, il faut que :

- Les interruptions soient activées. Elles sont activées/désactivées par les fonctions `Interrupt()`/`noInterrupts()`. Par défaut, elles sont activées dans le firmware Arduino.  
*Attention, les fonctions `attachInterrupt()` et `detachInterrupt()` ne sont utilisées que pour des interruptions externes sur les broches (i.e. INTERRUPTIONS MATERIELLES).*
- Le masque d'interruption soit activé. Il est activée/désactivée en définissant/effacer les bits du registre du masque d'interruption (`TIMSKx`).

*Dans la pratique: lorsqu'une interruption se produit, un drapeau « flag » est mis dans le registre d'interruption (`TIFRx`). Cette interruption est alors automatiquement effacée en entrant dans l'ISR ou en effacant manuellement le bit dans le registre contenant le flag.*

#### Exemples d'interruptions:

**1 - Timer Overflow:** se déclenche lorsque que le timer atteint sa valeur limite. Quand cela se produit, le « Timer Overflow bit » `TOVx` est mis dans le registre « timer interrupt flag register » `TIFRx`. Lorsque le « timer overflow interrupt enable bit » `TOIEx` est mis dans le registre « interrupt mask register » `TIMSKx`, le « timer overflow interrupt service routine » `ISR(TIMERx_OVF_vect)` est appelée.

**2 - Output Compare Match:** Quand un évènement Output Compare Match se produit, un flag `OCFxy` est mis dans le registre «interrupt flag register » `TIFRx`. Lorsque le “output compare interrupt enable bit” `OCIExy` est mis dans le registre “interrupt mask register `TIMSKx`”, l’“output compare match interrupt service” `ISR(TIMERx_COMPy_vect)` est appelée.

**3 – PWM:** utilise les registres `OCRxA`, `OCRxB`...

...

# EXEMPLE D'INTERRUPTION LOGICIELLE

## FAIRE CLIGNOTER UNE LED AVEC TIMER/COUNTER COMPARE MATCH INTERRUPT:

Cet exemple utilise le timer1 (16-bits) en mode CTC et le « compare match interrupt» pour faire clignoter une LED.

Le timer est configuré pour une fréquence de 2Hz. La LED est alors activée dans la routine de service d'interruption.

Données: fréquence d'horloge système = 16MHz, Prescaler à 256

Même exemple mais avec le TIMER OVERFLOW:

=> compteur= 65536 - (16MHz/(256\*65536))/2Hz=34286

```
#define ledPin 13

void setup()
{
  pinMode(ledPin, OUTPUT);

  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;

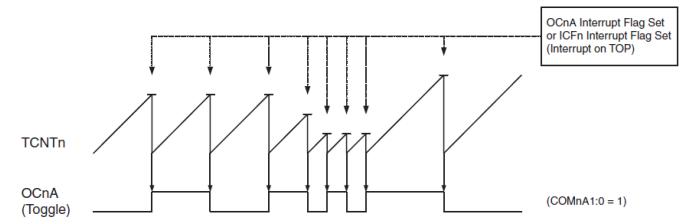
  TCNT1 = 34286;            // preload timer 65536-16MHz/256/2Hz
  TCCR1B |= (1 << CS12);   // 256 prescaler
  TIMSK1 |= (1 << TOIE1);  // enable timer overflow interrupt
  interrupts();              // enable all interrupts
}

ISR(TIMER1_OVF_vect)        // interrupt service routine that wraps
                            // a user defined function supplied by attachInterrupt
{
  TCNT1 = 34286;            // preload timer
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1);
}

void loop()
{
  // your program here...
}
```

<http://letsmakerobots.com/node/28278>

Figure 16-6. CTC Mode, Timing Diagram



=> compteur= (16MHz/(256\*65536))/2Hz=31250

```
#define ledPin 13

void setup()
{
  pinMode(ledPin, OUTPUT);

  // initialize timer1
  noInterrupts();           // disable all interrupts
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;

  OCR1A = 31250;            // compare match register 16MHz/256/2Hz
  TCCR1B |= (1 << WGM12);  // CTC mode
  TCCR1B |= (1 << CS12);   // 256 prescaler
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  interrupts();              // enable all interrupts
}

ISR(TIMER1_COMPA_vect)      // timer compare interrupt service routine
{
  digitalWrite(ledPin, digitalRead(ledPin) ^ 1); // toggle LED pin
}

void loop()
{
  // your program here...
}
```

Avec l'engouement pour les cartes Arduino, il était normal qu'une version d'un RTOS apparaisse: DuinOS

DuinOS est un système d'exploitation temps réel avec un système multitâche préemptif pour Arduino et Atmel AVR. Il est développé par RobotGroup (des argentins) et est basé sur FreeRTOS

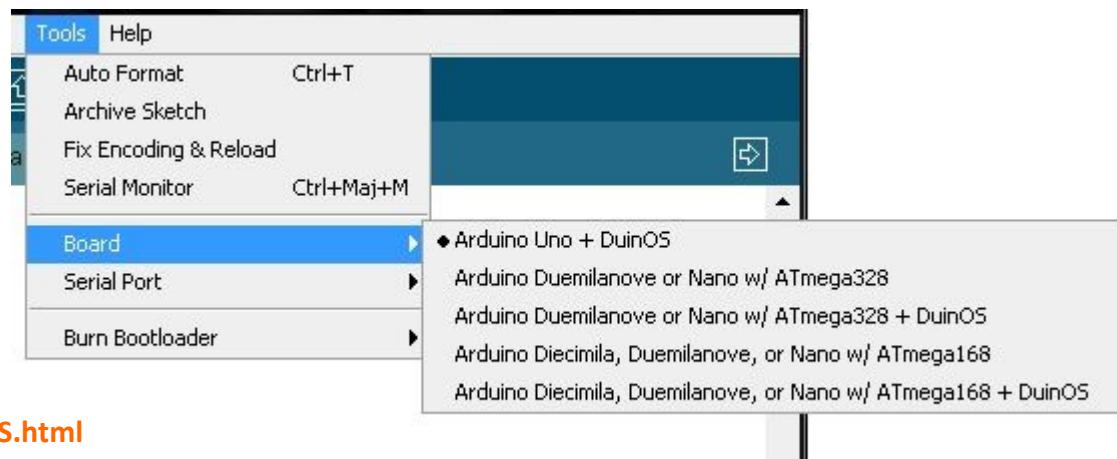
FreeRTOS est Système d'exploitation temps réel pour systèmes embarqués très populaire (portés sur plus de 30 microcontrôleurs). Il est distribué sous la licence GPL avec une exception facultative.

*L'exception autorise le code des utilisateurs de demeurer propriétaire des sources fermées, tout en conservant le noyau lui-même en open source, ce qui facilite l'utilisation de FreeRTOS dans des applications propriétaires. [2]*

Je vous propose d'installer DuinOS et d'écrire votre premier programme multi-tâches sous cet RTOS.

### Après l'installation de l'OS:

Une fois lancé l'environnement Arduino (avec le fichier arduino.exe situé à la racine), vous n'observerez pas de changement majeur : mais allez dans le menu "tools" puis le sous-menu "boards" et vous verrez les cartes en version DuinOS :



<http://www.freertos.org/>

<https://github.com/carlosdelfino/DuinOS>

<http://www.pobot.org/Premiers-pas-avec-DuinOS.html>

<http://code.google.com/p/duinos/>

Code pour faire simplement clignoter les deux leds en parallèle :

Schéma de connexion pour test rapide:

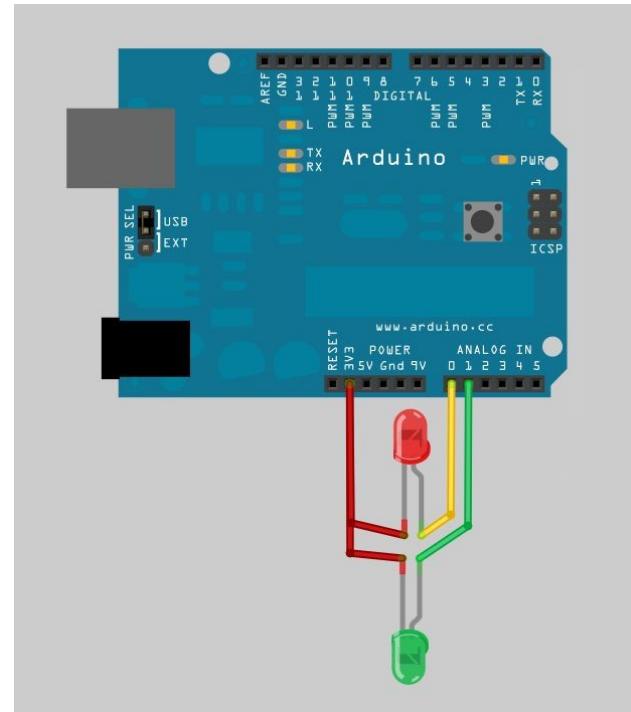
Connecter deux leds directement sur l'Arduino entre:

1 - le 3,3 volts (la tension la plus basse à disposition, et c'est déjà trop) et

2 - une patte contrôlable par une sortie « numérique ».

Rappel: "analog 0" et "analog 1" peuvent être utilisées simplement en tout ou rien, sous le nom de "digital 14" et "digital 15".

Peut-on mettre le pull-up ?



Si vous compilez en ayant sélectionné une carte qui n'a pas la mention "DuinOS", vous constaterez une erreur dès l'instruction "taskLoop" puisque le compilateur s'attend à voir une déclaration de fonction, or il n'y a pas de type devant.

error: expected constructor, destructor, or type conversion before '(' ...

```

/***
 * Exemple simple d'utilisation de DuinOS
 * basé sur l'exemple fourni dans la v0.2 "MoreComplexBlinking"
 * Il s'agit tout simplement de faire clignoter deux leds en parallèle à des rythmes différents.
 *
 * Based on the original Blink code by David Cuartielles
 *
 */

// on place les leds entre le 3,3 volts et les pattes analog 0 et 1 soit digital 14 et 15
int ledPinRed = 14;
int ledPinGreen = 15;

// Déclaration des deux tâches parallèles

taskLoop(redLED)
{
    digitalWrite(ledPinRed,HIGH);
    delay(200);
    digitalWrite(ledPinRed,LOW);
    delay(200);
}

taskLoop(greenLED)
{
    digitalWrite(ledPinGreen, HIGH);
    delay(500);
    digitalWrite(ledPinGreen, LOW); // set the LED off
    delay(500);
}

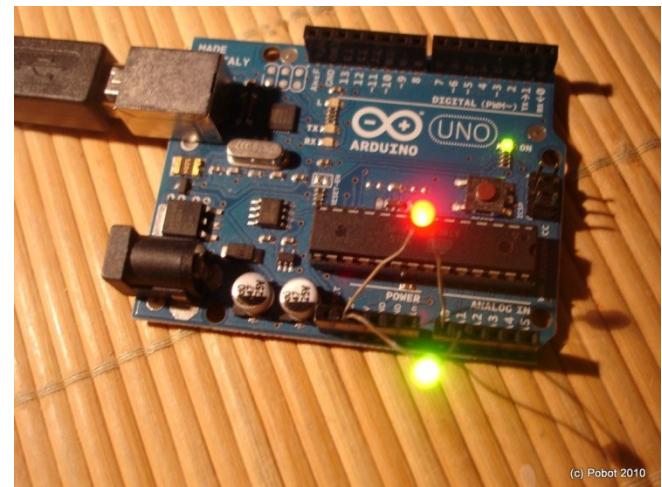
// Fonctions habituelles

void setup()
{
    // Initialisation des deux sorties pour les leds
    pinMode(ledPinRed, OUTPUT);
    pinMode(ledPinGreen, OUTPUT);
}

```

```

void loop()
{
    // Deux tâches en parallèle pendant 2 secondes
    resumeTask(greenLED);
    resumeTask(redLED);
    delay(2000);
    // Puis seulement la verte pendant 2 secondes
    suspendTask(redLED);
    delay(2000);
    // Puis seulement la rouge
    resumeTask(redLED);
    suspendTask(greenLED);
    delay(2000);
    // et on recommence (loop)
}
|
```



*Bien entendu, le CPU ne peut traiter qu'un ordre à la fois, donc le temps réel est obtenu en mettant un algorithme qui distribue le temps CPU tour par tour (de manière intelligente).*

*Vous savez au moins que vous avez un environnement qui fonctionne avec DuinOS, et vous pouvez sereinement commencer à découvrir la programmation multi-tâche temps réel.*

## LES PONTS-H

## COMMENT COMMANDER UN MOTEUR À COURANT CONTINU SACHANT QUE :

**Problème 1:** que l'on veut faire varier sa vitesse. Or, pour démarrer, un moteur à besoin d'une tension minimale et sa fréquence de rotation n'est pas réellement proportionnelle à la tension d'alimentation.

**Problème 2:** le moteur doit pouvoir tourner dans les deux sens et fonctionner dans les 4 quadrants, c'est-à-dire, soit en moteur (M), soit en génératrice (G) (freinage du moteur),

**Problème 3:** le courant absorbé est beaucoup plus important que ce que peut fournir un microcontrôleur.

Les solutions suivantes peuvent être envisagées (ce ne sont sûrement pas les seules)

### 1°) Résolution du problème 1 :

Il suffit de fournir au moteur une tension qui reste constante : la tension maximale et en même temps, cette tension ne sera appliquée que par très courtes périodes de temps => utilisation de la PWM  
En ajustant la durée de ces périodes, on arrive à faire varier la vitesse du moteur qui devient proportionnelle à la longueur des périodes de temps passées à la tension maximale par rapport au temps passé sans application de tension (tension nulle).

Problème, avec une PWM on ne peut contrôler un moteur DC que dans un seul sens .

Il serait possible d'utiliser un relai pour inverser la connexion (polarisation) du moteur mais cette option serait extrêmement gourmande en énergie.

### 2°) Résolution du problème 2: => le pont H.

[http://wiki.mdl29.net/doku.php?id=elec:moteur\\_a\\_courant\\_continu](http://wiki.mdl29.net/doku.php?id=elec:moteur_a_courant_continu)

# COMMANDÉ DE MOTEURS DC

## 2°) Résolution du problème 2 et 3 : PONT H

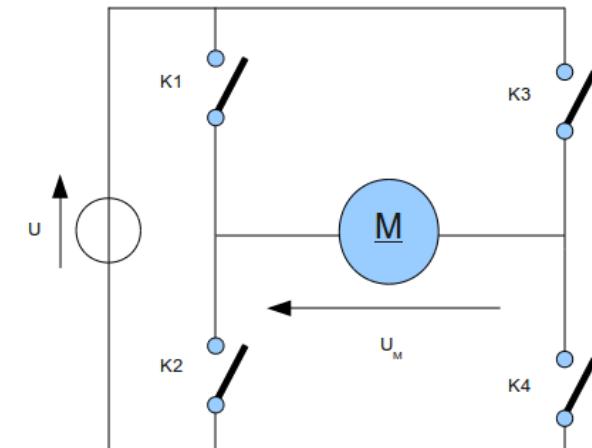
1 - Pour inverser le sens de rotation du moteur il suffit d'inverser la polarité de la tension à ses bornes. Le montage à utiliser est très simple : c'est un montage à 4 interrupteurs piloter 2 par 2 en alternance, d'où le nom pont en H.

Montés de telle façon, le courant peut passer soit dans un sens, soit dans l'autre dans la charge

### Fonctionnement :

K1 et K4 fermés, K2 et K3 ouverts :  $U_M = U$

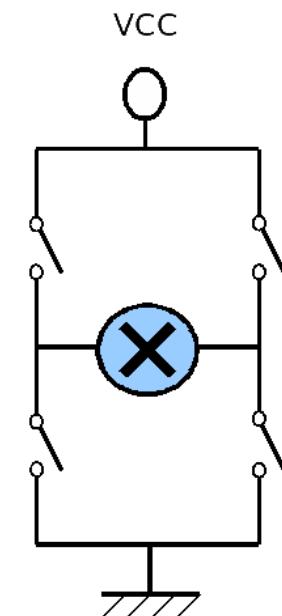
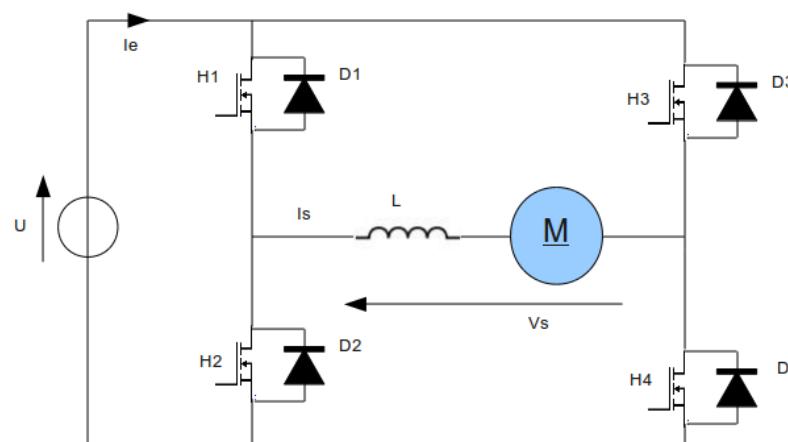
K1 et K4 ouverts, K2 et K3 fermés :  $U_M = -U$



2 - Les interrupteurs doivent pouvoir être commandés à l'ouverture et à la fermeture, et amplifier le courant fournit par le micro-contrôleur : on utilise des transistors en commutation , généralement des MOSFET.

### Montage :

$H_1, H_2, H_3$  et  $H_4$  sont des interrupteurs pilotés à l'ouverture et à la fermeture (ce peut être des transistors ou des thyristors). L'inductance permet de lisser le courant.



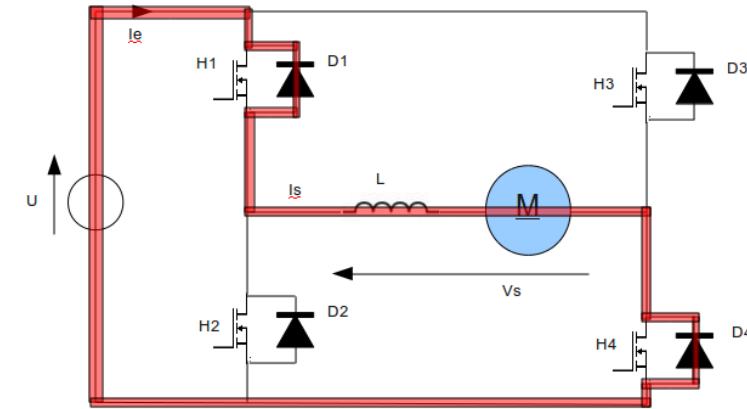
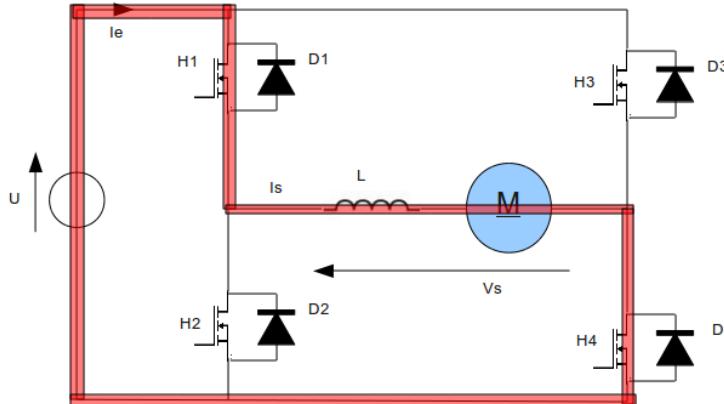
# COMMANDÉ DE MOTEURS DC

## Fonctionnement dans les 4 quadrants :

1)  $H_1$  et  $H_4$  fermés,  $H_2$  et  $H_3$  ouverts  $\Rightarrow V_s = U$  (le moteur tourne dans un sens)

*Si  $I_s > 0$  alors le courant passe par  $H_1$  et  $H_4$  (quadrant  $Q_1$ )*

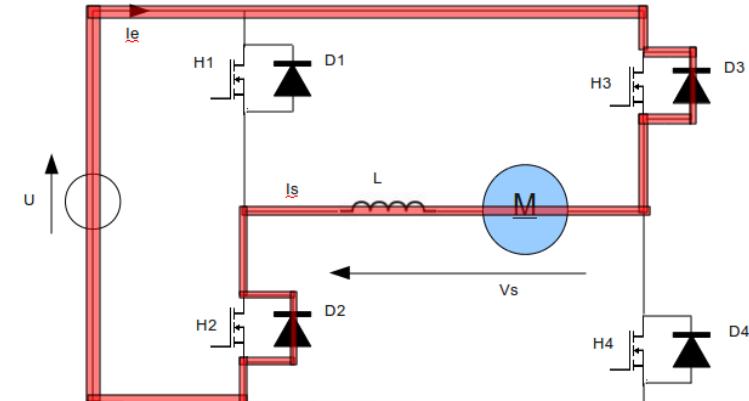
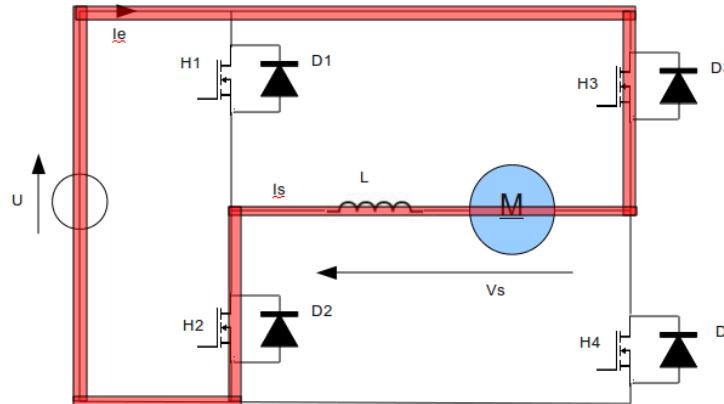
*Si  $I_s < 0$  alors le courant passe par  $D_1$  et  $D_4$  : (quadrant  $Q_2$ )*



2)  $H_1$  et  $H_4$  sont ouverts,  $H_2$  et  $H_3$  sont fermés  $\Rightarrow V_s = -U$  (le moteur tourne dans l'autre sens)

*Si  $I_s < 0$  alors le courant passe par  $H_2$  et  $H_3$  (quadrant  $Q_3$ )*

*Si  $I_s > 0$  alors le courant passe par  $D_2$  et  $D_3$  (quadrant  $Q_4$ )*



Les diodes sont appelées diodes de récupération, elles permettent la circulation du courant lorsque l'interrupteur est commandé et que le courant est dans le sens opposé à celui de l'interrupteur. Cette phase est appelée phase de récupération, elle correspond au freinage du moteur (qui fonctionne à ce moment précis en génératrice) appelé « freinage par récupération ».

# COMMANDE DE MOTEURS DC

Les L293NE et SN754410 sont des exemples basiques de pont-H:

**1 - Ils disposent de deux ponts:**

- l'un sur le côté gauche de la puce et
- l'autre sur la droite,

**2 - Ils peuvent commander 2 moteurs.**

**3 - Ils peuvent piloter jusqu'à 1A de courant**

**4 - et fonctionnent entre 4.5V et 36V.**

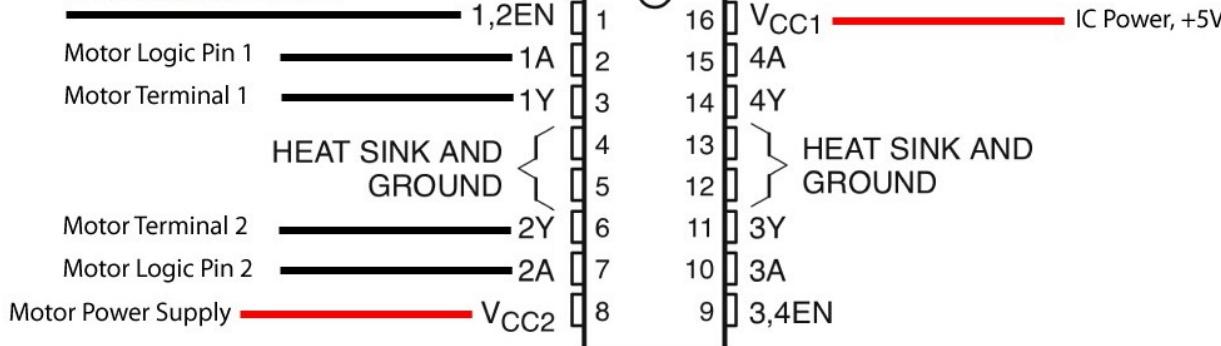


EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

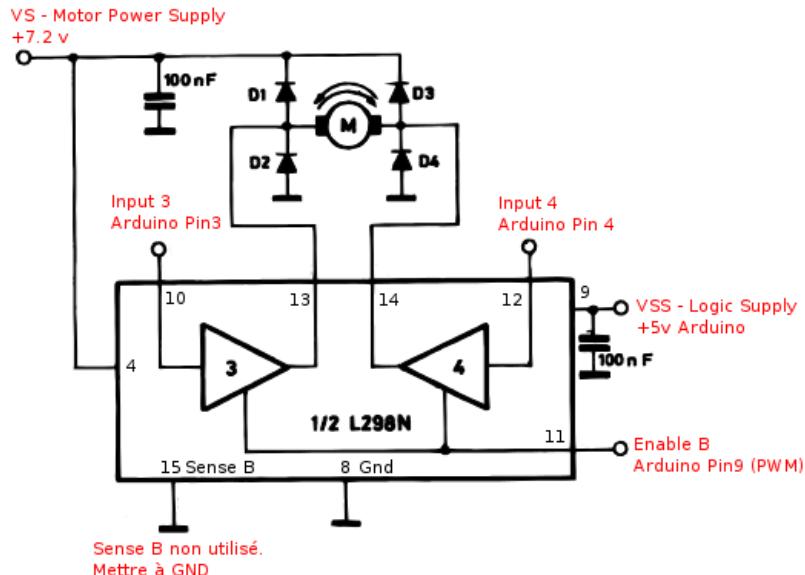
L293NE or SN754410

Connect to POWER to enable motor  
Connect to GROUND to disable motor



1 - Nécessite absolument 4 diodes en roue libre pour protéger les sorties (Output) du pont-H.

Utiliser des diodes de type "fast" ([BYV27-100-TAP: Diode Ultrafast, 2A, 100V. Farnell #1469371](#)).



2 - MOSFET Vs Bipolaires => Anticiper la chute de tension

Les transistors bipolaires commutés en saturation présentent une légère chute de tension ( $V_{CE\_SAT}$ ). Et comme le courant qui actionne le moteur passe par deux transistors, la chute de tension intervient donc deux fois!

Exemple L298N:

$V_{CEsat}(L)$	Sink Saturation Voltage	$I_L = 1A$ (5) $I_L = 2A$ (5)	0.85	1.2	1.6	$V$
$V_{CEsat}$	Total Drop	$I_L = 1A$ (5) $I_L = 2A$ (5)	1.80		3.2	$V$

Chute de tension  $V_{CE\_SAT}$  totale (total drop) de 1.8V (typique), 3.2V à  $I_L=1A$ , 4.9V à  $I_L=2A$ .

*Si cela semble beaucoup, c'est aussi un avantage car cela permet d'utiliser un accu de 7.2V directement avec un moteur 5V.*

# COMMANDE DE MOTEURS DC

```

int switchPin = 2;      // switch input
int motor1Pin1 = 3;     // pin 2 on L293D
int motor1Pin2 = 4;     // pin 7 on L293D
int enablePin = 9;      // pin 1 on L293D

void setup() {
    // set the switch as an input:
    pinMode(switchPin, INPUT);

    // set all the other pins you're using as outputs:
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(enablePin, OUTPUT);

    // set enablePin high so that motor can turn on:
    digitalWrite(enablePin, HIGH);
}

void loop() {
    // if the switch is high, motor will turn on one direction:
    if (digitalRead(switchPin) == HIGH) {
        digitalWrite(motor1Pin1, LOW);    // set pin 2 on L293D low
        digitalWrite(motor1Pin2, HIGH);   // set pin 7 on L293D high
    }
    // if the switch is low, motor will turn in the opposite direction:
    else {
        digitalWrite(motor1Pin1, HIGH);  // set pin 2 on L293D high
        digitalWrite(motor1Pin2, LOW);   // set pin 7 on L293D low
    }
}

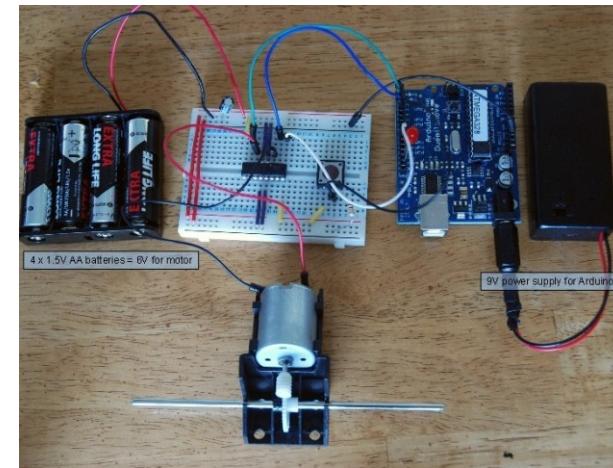
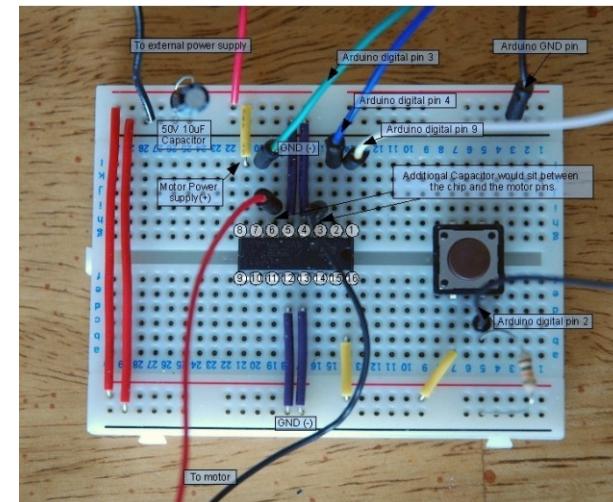
```

<http://luckyLarry.co.uk/arduino-projects/control-a-dc-motor-with-arduino-and-l293d-chip/>

[http://wiki.mdi29.net/doku.php?id=elec:moteur\\_a\\_courant\\_continu](http://wiki.mdi29.net/doku.php?id=elec:moteur_a_courant_continu)

<http://itp.nyu.edu/physcomp/Labs/DCMotorControl>

<http://arduino103.blogspot.com/2011/06/pont-h-transistor-pour-controler-un.html>



## III-7 LIAISONS SERIES: ASYNCHRONE & SYNCHRONE

# COMMUNICATIONS SERIES

## Communication Asynchrone



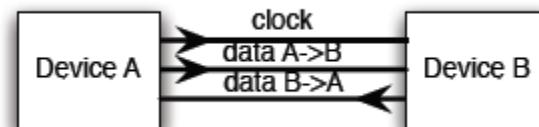
Asynchrone, car aucune Horloge.

Les données sont représentées en mettant HIGH/LOW à certains moments

Fils séparés pour la transmission et la réception

=> les deux dispositifs doivent communiquer au même rythme

## Communication Synchrone



Synchrone, car Horloge.

Les données sont représentées en mettant HIGH/LOW quand l'horloge change

Un seul fil d'horloge et de données pour chacune des directions

=> Pas besoin d'avoir deux dispositifs communiquant au même rythme, mais l'un des deux est le maître

## Quel est le meilleur ?

Cela dépend de votre application:

Les asynchrones sont bons lorsque qu'il n'y a que deux dispositifs et qu'ils sont pré-configurés pour parler à la même vitesse.

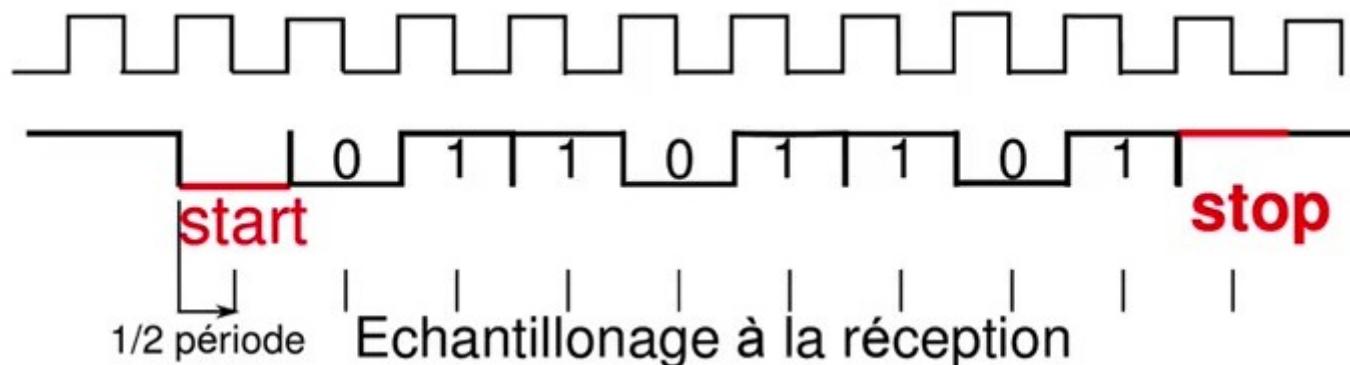
Les synchrones sont généralement meilleurs en vitesse

**LIAISON SERIE ASYNCHRONE:  
UART...**

Principe général:

On transmet sur une ligne 8 bits:

- 1 – précédés d'un start bit (mise à 0 sur une période) et
- 2 - suivi d'un silence (mise à 1 sur une période)



Valeurs usuelles: 8 bits, 1 stop bit, pas de parité, 9600 bits/s

Valeurs possibles : 300, ... 9600, 19200, ....115200 bits/s

**LIAISON SERIE SYNCHRONE:  
I<sup>2</sup>C, SPI, ONE WIRE, ...**

## LIAISON SERIE SYNCHRONE: I<sup>2</sup>C

Le bus I<sup>2</sup>C est caractérisé par une liaison en mode série réalisée à l'aide de 2 fils. C'est la société Philips qui en a créé le concept au début des années 80. Son succès est lié à sa simplicité.

Voici l'architecture type d'un bus I<sup>2</sup>C, on remarque que plusieurs composants (128 au max. (car adresse codage 7bits)) viennent se "greffer" sur le même bus,

Les données transitent par les lignes :

- SDA : signal de donnée, généré par le Maître ou l'Esclave.
- SCL : signal d'horloge généré par le Maître.

La communication sur le bus est orchestré de la manière suivante :

- Le **Maître** envoie sur le bus l'adresse du composant avec qui il souhaite communiquer,
- Chacun des esclaves ayant une adresse fixe, l'**esclave** qui se reconnaît répond à son tour par un signal de confirmation,
- Puis le **Maître** continue la procédure de communication...: (écriture/lecture)
- Dans tous les cas, les transactions seront confirmées par un **ACK (acquittement)**

Caractéristiques électriques :

Tension de bus : 5V DC

Fréquence maximale de fonctionnement : 400KHz (variable suivant les composants connectés au bus, voir datasheet)

Etats logiques: - Haut : de 3 à 5 Volts, - Bas : de 0 à 1.5 Volts

Capacité maximum admissible sur le bus : 400 pf

Temps de monté des signaux : inférieur à 1 µs

Temps de stabilité pour prise en compte d'un signal : supérieur à 5 µs (règle générale)

Résistance de rappel : à calculer en fonction de l'impédance du Bus (valeurs pratiques constatées de 1,5 K à 3,5 K)

Distance de communication : Quelque dizaines de centimètres, et plusieurs mètres avec un tampon.

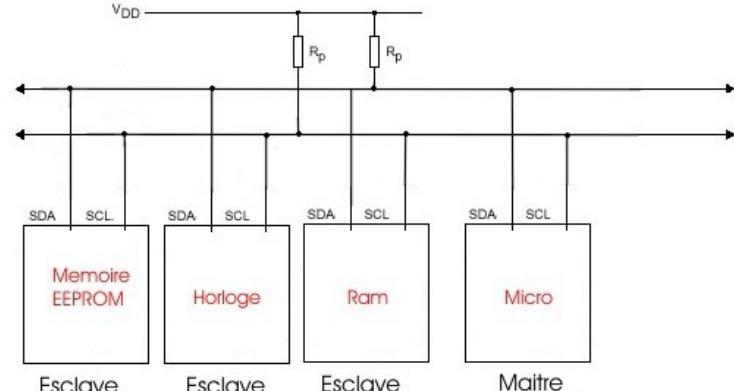
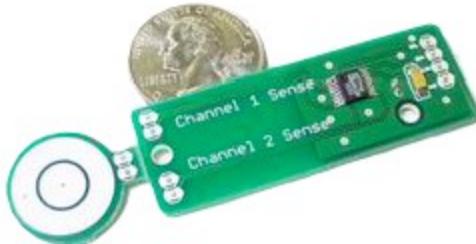
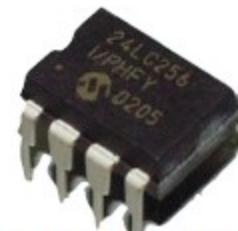


Schéma d'interface matérielle des circuits compatibles I<sup>2</sup>C

Exemples de composants utilisant l'I<sup>2</sup>C: des mémoires, convertisseurs analogiques/digitaux et digitaux /analogiques, des horloges temps réel, des synthétiseurs de fréquence intégrés pour récepteurs radio ou TV...



touch sensor



non-volatile  
memory



compass

*Boussole digitale  
indiquant le nord*



fm transmitter



LCD display

And many others  
(gyros, keyboards, motors,...)

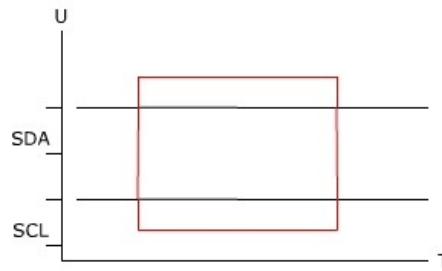


temperature &  
humidity sensor

# LES BASES DU PROTOCOLE

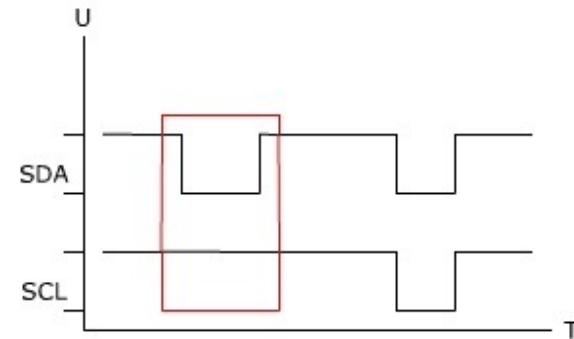
## 1- LA CONDITION DE REPOS :

Condition dans laquelle aucun composant ne communiquent, les lignes de bus sont à l'état Haut :



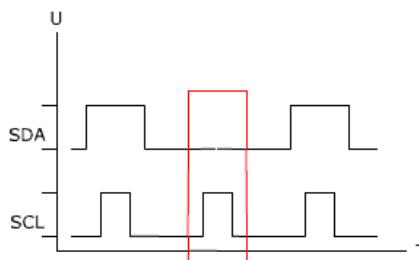
## 2- LA CONDITION DE DEPART :

Le **Maître** force la ligne SDA au niveau bas pour "réveiller" les **Esclaves** quand SCL est au niveau haut :



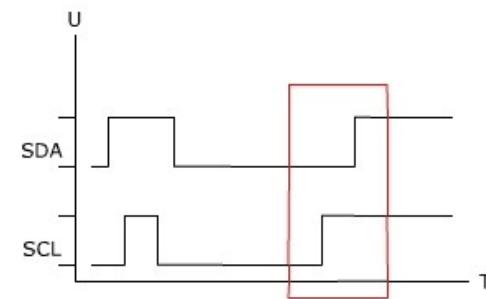
## 3- L'ACQUITTEMENT :

Une fois les données transmises, un acquittement est opéré par celui qui reçoit les données. Cette procédure est réalisée en fin de trame de transmission, soit à la neuvième impulsion. Le récepteur maintient la ligne SDA au niveau bas pendant le front d'horloge :



## 4- LA CONDITION D'ARRET :

Les lignes SDA et SCL sont au niveau bas, puis quand SCL passe au niveau haut, le **Maître** libère la ligne SDA au niveau haut. A cet instant le bus est considéré comme disponible :



# ARCHITECTURE D'UNE TRAME

## LA TRAME:

1 - il faut réveiller le bus en réalisant une condition de départ,

2 - ensuite viennent s'intercaler les trames de données.

**Le bit de poids fort étant envoyé en premier.** L'**Esclave** ayant reçu les informations acquittera le **Maître** pour lui faire savoir qu'il a bien reçu la trame de données, dans le cas contraire, c'est au maître de recommencer la transaction depuis le départ.

3 - Pour finir le signal de Stop mettra fin à toute communication.

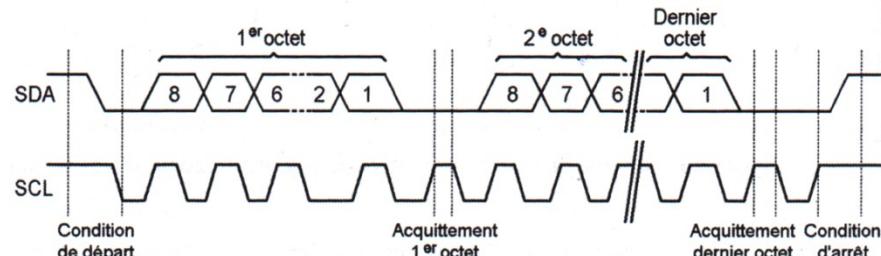
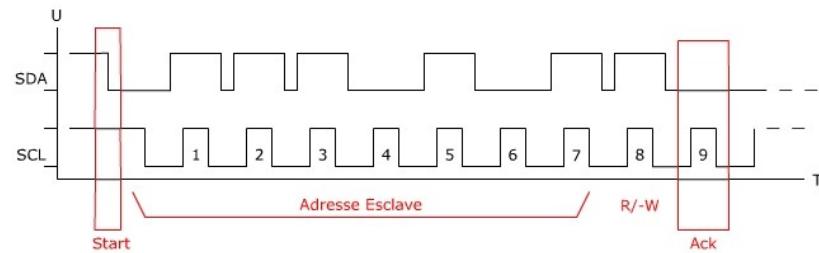


Figure 6.8 – Chronogramme d'un échange complet sur le bus I2C.

**L'ADRESSAGE :** pour communiquer avec les ≠ composants raccordés au bus,

1 - Après avoir émis une condition de départ, il faut que le **Maître** indique avec quel **Esclave** il souhaite se connecter, pour cela il enverra sur le bus l'adresse de l'**Esclave** composé de 7 bits,

2 - puis un dernier indiquant le sens du transfert : **Lecture** (1: esclave->maître) ou **Écriture** (0 : maître->esclave).



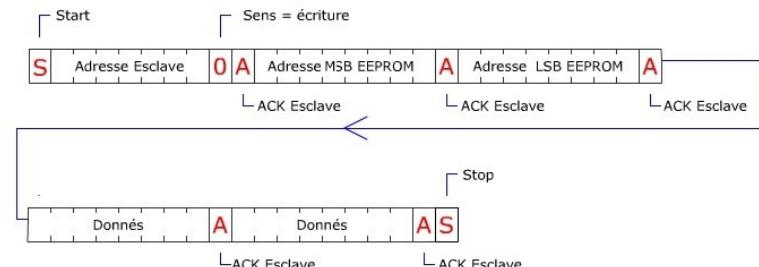
L'**esclave** reconnaissant son adresse validera par un acquittement...

Les fonctions de la librairie Wire: <http://www.arduino.cc/en/Reference/Wire>

## LA COMMUNICATION PAR OCTET : un échange complet sur le bus entre Maitre et Esclave

### 1 - Séquence d'**ÉCRITURE** de données dans une mémoire EEPROM I<sup>2</sup>C :

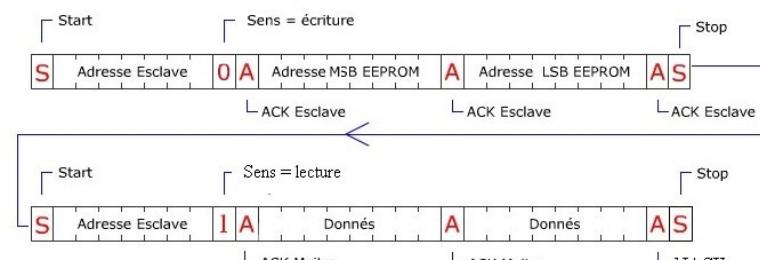
- Condition de départ
- Emission de l'adresse esclave, soit de L'EEPROM ici
- Acquittement de la part de l'esclave
- Emission par le Maître de l'adresse mémoire à laquelle il souhaite inscrire les futures données, acquittement successif de l'esclave
- Transmission des données à écrire, acquittement à chaque réception de paquet par l'esclave
- Condition de stop



On remarque que le Maître, après avoir émis l'adresse de stockage des données, n'a pas besoin d'interrompre la communication à chaque octet et d'écrire à nouveau l'adresse de pointage. En règle générale les composants I<sup>2</sup>C sont à 'auto-chargement', c'est à dire que leur pointeur d'adresse s'incrémentent automatiquement en fin d'émission et ce pour la valeur de 8 octets.

### 2 - Séquence de **LECTURE** de données dans une mémoire EEPROM I<sup>2</sup>C :

- 1 - Condition de départ
- Emission de l'adresse esclave, soit de L'EEPROM, **avec une demande d'écriture**. Pourquoi ? simplement pour faire pointer le registre interne de la mémoire sur l'adresse choisie, sans cela, la lecture se ferait à la valeur contenue dans ce pointeur, ce qui ne vous retournerez pas les bonnes valeurs.
- Acquittement de la part de l'esclave et condition de stop (Restart est possible)
- 2 - Condition de départ
- Emission de l'adresse esclave, soit de L'EEPROM, **avec une demande de lecture cette fois**. Le composant confirme l'ordre et les données sont envoyées. La lecture peut se faire par bloc, dans ce cas, il faut **confirmer chaque donnée lu par un ACK, puis avant la condition de stop, envoyé un NACK**.
- Condition de stop



# REMARQUES: LES ADRESSES ESCLAVES

A chaque composant qui constitue le bus I<sup>2</sup>C est attribué une adresse,  
MAIS QUE FAIRE SI IL Y AVAIT PLUSIEURS FOIS LES MÊMES TYPES DE COMPOSANTS PRÉSENTS SUR LE BUS ?

Pour généraliser, chaque famille de circuit est composée d'une adresse «silicium», indélébile et non modifiable, puis d'une autre partie entièrement configurable de manière externe, totalement accessible au niveau des bornes du circuit intégré :

Dans l'exemple ci-dessous, l'adresse du PCF8583 (Horloge-calendrier + RAM) est composée de 6 bits fixes et 1 modifiable, soit la possibilité de brancher 2 circuits identiques sur le bus.

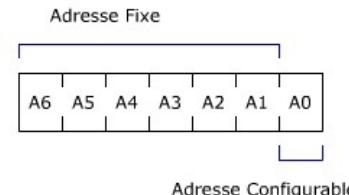
Table 8. SDA Write Mode Bit Format

S	0	1	0	1	1	0	AD0	0	A
Slave Address Byte									

Composant I<sup>2</sup>C les plus utilisés ainsi que la composition de leurs adresses fixes :

Type	Description
PCD 3311	Générateur / Décodeur DTMF
PCF 8570	RAM Statique 2 Ko (256 x 8)
PCF 8573	Horloge/Calendrier temps réel
PCF 8574	Extension de port parallèle 8 Bits
PCF 8552	EEprom 2 Ko (256 x 8)
PCF 8583	Horloge/Calendrier temps réel + 240 Octets de RAM
PCF 8591	Convertisseur A/N N/A 8 Bits
SAA 1064	Drivers d'affichage à leds 4 Digits
M24C256	EEprom 128 Ko (32k x 8)

## Adresse Fixe



Exemple : PCF8583

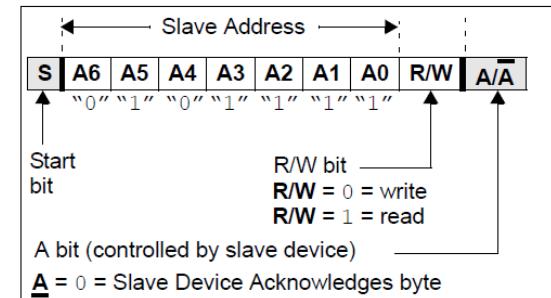
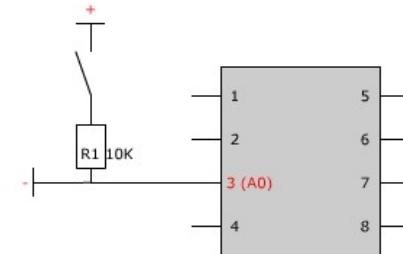


FIGURE 5-9: Slave Address Bits in the I<sup>2</sup>C Control Byte.

On remarque que certains circuits possèdent la même adresse, mais il s'agit uniquement de composants de fonctions similaires comme les mémoires...

# EXEMPLE DE PROGRAMME

Joindre le bus I<sup>2</sup>C  
(en tant que Maître)



```
#include <Wire.h>

void setup() {
    Wire.begin();          // join i2c bus (address optional for master)
    Serial.begin(9600);   // start serial for output
}

void loop() {
    Wire.requestFrom(2, 6);    // request 6 bytes from slave device #2

    while(Wire.available()) {  // slave may send less than requested
        char c = Wire.receive(); // receive a byte as character
        Serial.print(c);       // print the character
    }

    delay(500);
}
```

Demander les données du  
dispositif



Obtenir les données



# EXEMPLE D'UTILISATION DU BUS I<sup>2</sup>C

Commande d'un potentiomètre numérique (AD5171) à interface I<sup>2</sup>C:

**Rappel:**

Pin I<sup>2</sup>C sur l'Arduino: SDA: Analog In 4, SCK: Analog In 5

```
#include <Wire.h>
byte position = 0;

void setup ()
{
  Wire.begin (); // join i2c bus, Arduino en Maître
}

void loop ()
{
  Wire.beginTransmission (0x2C); // L'AD5171 est à l'adresse 0x2C
  Wire.send (0x40);           // Envoi de l'instruction
  Wire.send (position);       // Envoi de la position
  Wire.endTransmission ();    // Fin de l'échange

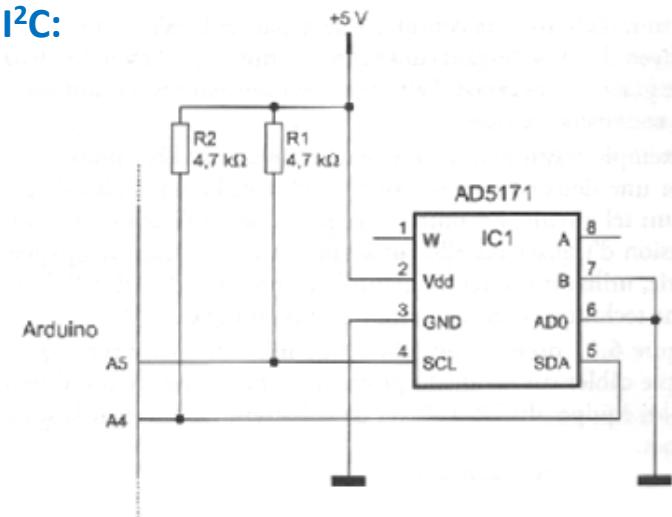
  Position++;
  if (position == 64) //si le maximum (64 positions pour AD5171) est atteint
  {
    Position=0;
  }
  Delays (500);
}
```

S	0	1	0	1	1	0	AD0	0	A
Slave Address Byte									

$$\begin{matrix} 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{matrix}$$

Adresse 44 en décimal ( $2^5 + 2^3 + 2^1 = 44$ ) => 0x2C en Héxadécimal

I<sup>2</sup>C Device Address Bit. Allows a maximum of two AD5171s to be addressed.



FUNCTIONAL BLOCK DIAGRAM

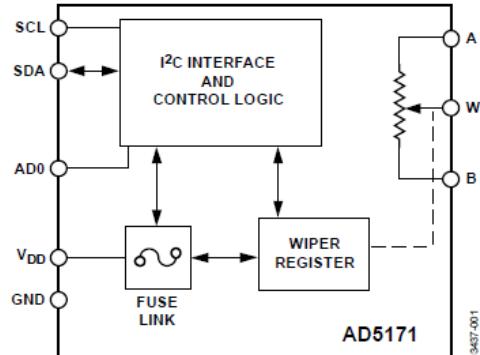


Figure 1.

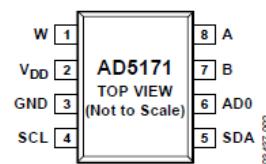
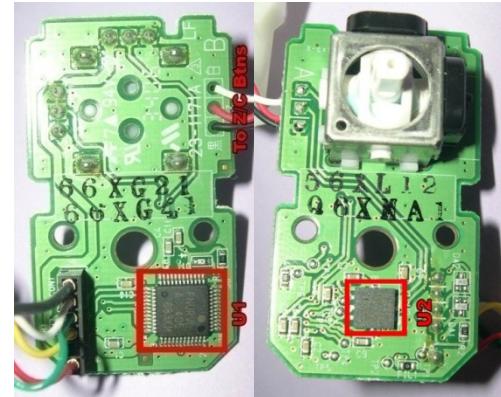


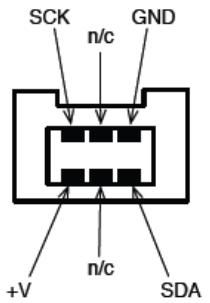
Figure 2. Pin Configuration

Le Nunchuk de la Wii possède:

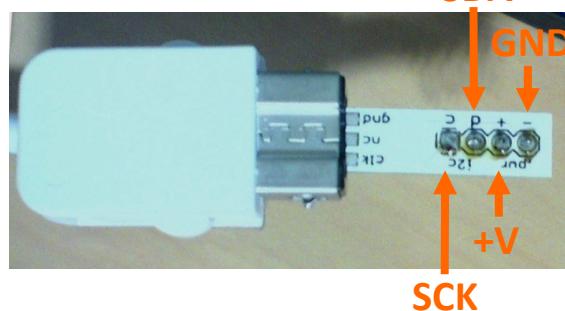
- 1 - En standard une interface I2C
- 2 - Un accéléromètre 3 axes de résolution 10bits.
- 3 - Un joystick analogique 2 axes avec un convertisseur A/D 8 bits
- 4 - 2 boutons



Nunchuk PinOUT



Adaptateur PinOUT



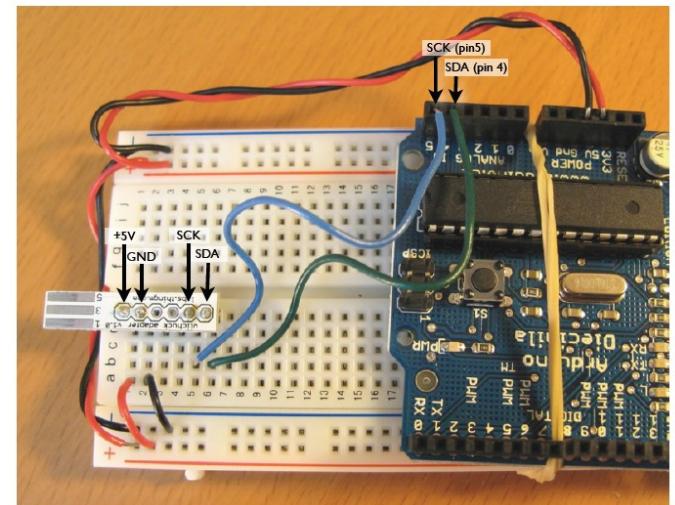
### Programme: Nunchuk Print

Lire les données toutes les 100ms, et afficher:

- Position du joystick (x,y)
- Accelerometre (x,y,z)
- Boutons Z,C

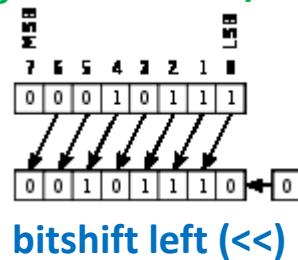
Chip listing:

U1: Unknown manufacturer - "FNURVL" "A 405" "633KM" (custom processing chip? - TQFP48)  
 U2: [ST Microelectronics LIS3L02AL 3-axis accelerometer - "1806" "3L02AE" "615" "MLT"](#) (accelerometer - LGA-8)

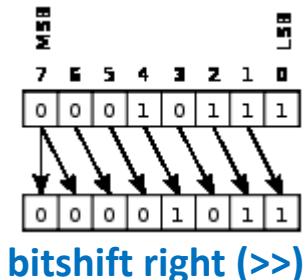


Il y a deux opérateurs de décalage de bits en C/C++:

1 - l'opérateur de décalage à gauche << (bitshift left) qui provoque le décalage à gauche des bits,



2 - l'opérateur de décalage à droite >> (bitshift right) qui provoque le décalage à droite des bits.



Syntaxe: variable << number\_of\_bits ou variable >> number\_of\_bits

Exemples:

int a = 5; // binary: 0000000000000101

int b = a << 3; // binary: 0000000000101000 => 40 en décimal ( $5 \times 2^3 = 5 \times 8 = 40$ )

int c = b >> 3; // binary: 0000000000000101 => on revient à 5

Remarque1 : quand vous décalez un nombre vers la gauche le bit le plus à gauche est littéralement perdu !

Exemple:

int a = 5; // binary: 0000000000000101

int b = a << 14; // binary: 0100000000000000 – le premier 1 du 101 est perdu !

Remarque2 : Le décalage d'un nombre n à gauche multiplie par  $2^n$

$1 << 0 == 1 = 2^0$

$1 << 1 == 2 = 2^1$

$1 << 2 == 4 = 2^2$

...

$1 << 8 == 256 = 2^8$

$1 << 10 == 1024 = 2^{10}$

...

Au contraire, le décalage d'un nombre n à droite divise par  $2^n$

# Bitwise AND (&), Bitwise OR (|), Bitwise XOR (^)

Les opérateurs bitwise (AND, OR, XOR) effectuent leurs calculs sur chaque position de bit.

## Bitwise AND (&):

Si les deux bits d'entrée sont à 1, la sortie est à 1, sinon à 0.

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 0 0 1 (operand1 & operand2) - returned result

Une des utilisations les plus courantes du AND est de sélectionner des bits par « masquage ».

## Bitwise OR (|):

Le OU binaire de deux bits est 1 si l'un OU l'autre des bits d'entrée est à 1, sinon il vaut 0.

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 1 1 1 (operand1 | operand2) - returned result

Bitwise XOR (^): Cet opérateur est très similaire à l'opérateur OR (|), seulement il met un bit à 0 lorsque les deux bits d'entrée pour sont 1:

0 0 1 1 operand1

0 1 0 1 operand2

-----

0 1 1 0 (operand1 ^ operand2) - returned result

## APPLICATIONS: décodage des informations du Nunchuk

```
// byte outbuf[5] contains bits for z and c buttons and it also contains the least significant bits for the  
accelerometer data  
// so we have to check each bit of byte outbuf[5]  
if ((outbuf[5] >> 0) & 1) { z_button = 1; }  
if ((outbuf[5] >> 1) & 1) { c_button = 1; }  
if ((outbuf[5] >> 2) & 1) { accel_x_axis += 2; }
```

<http://todbot.com/blog/bionicarduino/>

```
#include <Wire.h>
#include <string.h>
#undef int
#include <stdio.h>

uint8_t outbuf[6]; // array to store arduino output
int cnt = 0;
int ledPin = 13;

void setup ()
{
beginSerial (19200);
Serial.print ("Finished setup\n");
Wire.begin (); // join i2c bus as master
nunchuck_init (); // send the initialization handshake
}

void nunchuck_init ()
{
Wire.beginTransmission (0x52); // transmit to device 0x52
Wire.send (0x40); // sends memory address
Wire.send (0x00); // sends sent a zero.
Wire.endTransmission (); // stop transmitting
}

void send_zero ()
{
Wire.beginTransmission (0x52); // transmit to device 0x52
Wire.send (0x00); // sends one byte
Wire.endTransmission (); // stop transmitting
}

Void loop ()
{
Wire.requestFrom (0x52, 6); // request data from nunchuck
while (Wire.available ())
{
outbuf[cnt] = nunchuk_decode_byte (Wire.receive()); // receive byte as an integer
digitalWrite (ledPin, HIGH); // sets the LED on
cnt++;
}
}
```

```
// If we received the 6 bytes, then go print them
if (cnt >= 5)
{
print ();
}
cnt = 0;
send_zero (); // send the request for next bytes
delay (100);
}

// Print the input data we have received
// accel data is 10 bits long
// so we read 8 bits, then we have to add
// on the last 2 bits. That is why I
// multiply them by 2 * 2
void print ()
{
int joy_x_axis = outbuf[0];
int joy_y_axis = outbuf[1];
int accel_x_axis = outbuf[2] * 2 * 2;
int accel_y_axis = outbuf[3] * 2 * 2;
int accel_z_axis = outbuf[4] * 2 * 2;

int z_button = 0;
int c_button = 0;

// byte outbuf[5] contains bits for z and c buttons
// it also contains the least significant bits for the
accelerometer data
// so we have to check each bit of byte outbuf[5]
if ((outbuf[5] >> 0) & 1)
{
z_button = 1;
}
if ((outbuf[5] >> 1) & 1)
{
c_button = 1;
}

if ((outbuf[5] >> 2) & 1)
{
accel_x_axis += 2;
}
if ((outbuf[5] >> 3) & 1)
{
accel_x_axis += 1;
}}
```

<http://www.windmeadow.com/node/42>

```
if ((outbuf[5] >> 4) & 1)
{
accel_y_axis += 2;
}

if ((outbuf[5] >> 5) & 1)
{
accel_y_axis += 1;
}

if ((outbuf[5] >> 6) & 1)
{
accel_z_axis += 2;
}
if ((outbuf[5] >> 7) & 1)
{
accel_z_axis += 1;
}

Serial.print (joy_x_axis, DEC);
Serial.print ("\t");
Serial.print (joy_y_axis, DEC);
Serial.print ("\t");
Serial.print (accel_x_axis, DEC);
Serial.print ("\t");
Serial.print (accel_y_axis, DEC);
Serial.print ("\t");
Serial.print (accel_z_axis, DEC);
Serial.print ("\t");
Serial.print (z_button, DEC);
Serial.print ("\t");
Serial.print (c_button, DEC);
Serial.print ("\t");
Serial.print ("\r\n");

// Encode data to format that most wiimote drivers except
// only needed if you use one of the regular wiimote drivers
char nunchuk_decode_byte (char x)
{
x = (x ^ 0x17) + 0x17;
return x;
}
```



RETOUR

L'Arduino doit envoyer un HandShake pour communiquer avec le Nunchuck.

Ainsi, envoyez premièrement 2 bytes "0x40, 0x00".

Puis envoyez seulement 1 byte "0x00" à chaque fois que vous sollicitez les données du Nunchuck.

Ces données vous seront renvoyées par paquets de 6 octets.

Byte	Description	Values of sample Nunchuk
1	X-axis value of the analog stick	Min(Full Left):0x1E / Medium(Center):0x7E / Max(Full Right):0xE1
2	Y-axis value of the analog stick	Min(Full Down):0x1D / Medium(Center):0x7B / Max(Full Right):0xDF
3	X-axis acceleration value	Min(at 1G):0x48 / Medium(at 1G):0x7D / Max(at 1G):0xB0
4	Y-axis acceleration value	Min(at 1G):0x46 / Medium(at 1G):0x7A / Max(at 1G):0xAF
5	Z-axis acceleration value	Min(at 1G):0x4A / Medium(at 1G):0x7E / Max(at 1G):0xB1
6	Button state (Bits 0/1) / acceleration LSB	Bit 0: "Z"-Button (0 = pressed, 1 = released) / Bit 1: "C" button (0 = pressed, 1 = released) / Bits 2-3: X acceleration LSB / Bits 4-5: Y acceleration LSB / Bits 6-7: Z acceleration LSB

#### Data byte receive

Joystick X

Joystick Y

Accelerometer X (bit 9 to bit 2 for 10-bit resolution)

Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)

Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)

Accel. Z bit 1	Accel. Z bit 0	Accel. Y bit 1	Accel. Y bit 0	Accel. X bit 1	Accel. X bit 0	C-button	Z-button
-------------------	-------------------	-------------------	-------------------	-------------------	-------------------	----------	----------

Octet 1 : Joystick - Axe X : un octet donc de 0 à 255

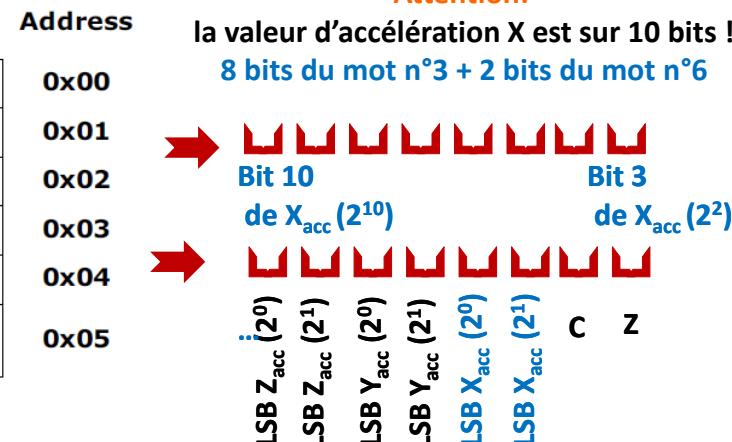
Octet 2 : Joystick - Axe Y : un octet donc de 0 à 255

Octet 3 : Accéléromètre - Axe X : un octet représentant les 8 bits de poids fort sur les 10 bits possibles, si on utilise que cet octet on aura des valeurs de 0 à 255

Octet 4 : Accéléromètre - Axe Y : 8 bits de poids fort sur les 10 bits possibles : de 0 à 255

Octet 5 : Accéléromètre - Axe Z : 8 bits de poids fort sur les 10 bits possibles : de 0 à 255

Octet 6 : du MSB au LSB : 2 bits pour l'axe Z de l'accéléromètre, concaténé avec les 8 bits précédents vous pouvez obtenir 10 bits de résolution soit des valeurs de 0 à 1023. 2 bits pour l'axe Y de l'accéléromètre, 2 bits pour l'axe X de l'accéléromètre. 1 bit qui représente l'état du bouton C : 0:enfoncé, 1:relâché. 1 bit qui représente l'état du bouton Z : 0:enfoncé, 1:relâché.



```

/* PROGRAMME PRINCIPAL
*/

#include <math.h>
#include <Servo.h>
#include "Wire.h"
#include "WiiChuck.h"
#define MAXANGLE 90
#define MINANGLE -90

Servo myservo; // create servo object to control a servo
int potpin = 0; // analog pin used to connect the potentiometer

WiiChuck chuck = WiiChuck();

int angleStart, currentAngle;
int tillerStart = 0;
double angle;

void setup()
{
    //nunchuck_init();
    Serial.begin(115200);
    chuck.begin();
    chuck.update();
    //chuck.calibrateJoy();
    // attaches the servo on pin 9 to the servo object
    myservo.attach(9);
}

void loop()
{
    delay(20);
    chuck.update();
    // sets the servo position according to the scaled value
    myservo.write(chuck.readPitch());
    Serial.print(chuck.readRoll());
    Serial.print(", ");
    Serial.print(chuck.readPitch());
    Serial.print(", ");

    Serial.print((int)chuck.readAccelX());
    Serial.print(", ");
    Serial.print((int)chuck.readAccelY());
    Serial.print(", ");
    Serial.print((int)chuck.readAccelZ());
    Serial.println();
}

```

## Insertion de la librairie: WiiChuck.h (cf transparent suivant)



# Nintendo Wii Nunchuck & SERVO (Version Library)

```

/*
 * Nunchuck -- Use a Wii Nunchuck
 * Tim Hirzel http://www.growdown.com
 *
notes on Wii Nunchuck Behavior.
This library provides an improved derivation of rotation angles from the nunchuck accelerometer
data. The biggest different over existing libraries (that I know of) is the full 360 degrees of Roll data
from teh combination of the x and z axis accelerometer data using the math library atan2. It is
accurate with 360 degrees of roll (rotation around axis coming out of the c button, the front of the
wii), and about 180 degrees of pitch (rotation about the axis coming out of the side of the wii).
(read more below)
In terms of mapping the wii position to angles, its important to note that while the Nunchuck
sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an important
disparity where the nunchuk only works within one hemisphere. At a result, when the pitch
values are less than about 10, and greater than about 170, the Roll data gets very unstable.
essentially, the roll data flips over 180 degrees very quickly. To understand this property better,
rotate the wii around the axis of the joystick. You see the sensor data stays constant (with noise).
Because of this, it cant know the difference between arriving upside via 180 degree Roll, or 180
degree pitch. It just assumes its always 180 roll.
*/
* This file is an adaptation of the code by these authors:
* Tod E. Kurt, http://todbot.com/blog/
* The Wii Nunchuck reading code is taken from Windmeadow Labs
* http://www.windmeadow.com/node/42
*/
#ifndef WiiChuck_h
#define WiiChuck_h

#include "WProgram.h"
#include <Wire.h>
#include <math.h>

// these may need to be adjusted for each nunchuck for calibration
#define ZEROX 510
#define ZEROY 490
#define ZEROZ 460
#define RADIUS 210 // probably pretty universal

#define DEFAULT_ZERO_JOY_X 124
#define DEFAULT_ZERO_JOY_Y 132

class WiiChuck {
private:
    byte cnt;
    uint8_t status[6];// array to store wiichuck output
    byte averageCounter;
    //int accelArray[3][AVERAGE_N]; // X,Y,Z
    int i;
    int total;
    uint8_t zeroJoyX; // these are about where mine are
    uint8_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct
    int lastJoyX;
    int lastJoyY;
    int angles[3];
    boolean lastZ, lastC;
public:
    byte joyX;
    byte joyY;
    boolean buttonZ;
    boolean buttonC;
    void begin()
    {
        Wire.begin();
        cnt = 0;
        averageCounter = 0;
        // instead of the common 0x40 -> 0x00 initialization, we use 0xF0 -> 0x55 followed
        // by 0xFB -> 0x00, this lets us use 3rd party nunchucks (like cheap $4 ebay ones)
        // while still letting us use official ones. Only side effect is that we no longer need
        // to decode bytes in _nunchuk_decode_byte
        // see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264805255

        Wire.beginTransmission(0x52);// device address
        Wire.send(0xF0);
        Wire.send(0x55);
        Wire.endTransmission();
        delay(1);
        Wire.beginTransmission(0x52);
        Wire.send(0xFB);
        Wire.send(0x00);
        Wire.endTransmission();
        update();
        for (i = 0; i<3;i++) {
            angles[i] = 0;
        }
        zeroJoyX = DEFAULT_ZERO_JOY_X;
        zeroJoyY = DEFAULT_ZERO_JOY_Y;
    }

    void calibrateJoy()
    {
        zeroJoyX = joyX;
        zeroJoyY = joyY;
    }

    void update()
    {
        Wire.requestFrom (0x52, 6); // request data from nunchuck
        while (Wire.available ()) { // receive byte as an integer
            status[cnt] = _nunchuk_decode_byte (Wire.receive()); //
            cnt++;
        }
        if (cnt > 5) {
            lastZ = buttonZ;
            lastC = buttonC;
            lastJoyX = readJoyX();
            lastJoyY = readJoyY();
            //averageCounter++;
            //if (averageCounter >= AVERAGE_N)
            //    averageCounter = 0;
            cnt = 0;
            joyX = (status[0]);
            joyY = (status[1]);
            for (i = 0; i < 3; i++)
                //accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2)));
                angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2) ) >> ((i+1)*2));
                //accelArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);
                //accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B11000000) >> 6);
                buttonZ = !(status[5] & B00000001);
                buttonC = !(status[5] & B00000010) >> 1;
                _send_zero(); // send the request for next bytes
        }
    }

    // UNCOMMENT FOR DEBUGGING
    //byte * getStatus()
    //{
    //    return status;
    //}
    float readAccelX()
    {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[0] - ZEROY;
    }
    float readAccelY()
    {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[1] - ZEROY;
    }
    float readAccelZ()
    {
        // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
        return (float)angles[2] - ZEROZ;
    }
    boolean zPressed()
    {
        return (buttonZ && ! lastZ);
    }
    boolean cPressed()
    {
        return (buttonC && ! lastC);
    }
    // for using the joystick like a directional button
    boolean rightJoy(int thresh=60)
    {
        return (readJoyX() > thresh and lastJoyX <= thresh);
    }
    // for using the joystick like a directional button
    boolean leftJoy(int thresh=60)
    {
        return (readJoyX() < -thresh and lastJoyX >= -thresh);
    }
    int readJoyX()
    {
        return (int)joyX - zeroJoyX;
    }
    int readJoyY()
    {
        return (int)joyY - zeroJoyY;
    }
    // R, the radius, generally hovers around 210 (at least it does with mine)
    int R()
    {
        // return sqrt(readAccelX() * readAccelX() +readAccelY() * readAccelY() +
        readAccelZ() * readAccelZ());
    }
    // returns roll degrees
    int readRoll()
    {
        return (int)(atan2(readAccelX(),readAccelZ())/ M_PI * 180.0);
    }
    // returns pitch in degrees
    int readPitch()
    {
        return (int)(acos(readAccelY() / RADIUS)/ M_PI * 180.0); // optionally
        swap 'RADIUS' for 'R'
    }
private:
    byte _nunchuk_decode_byte (byte x)
    {
        //decode is only necessary with certain initializations
        //x = (x ^ 0x17) + 0x17;
        return x;
    }
    void _send_zero()
    {
        Wire.beginTransmission (0x52); // transmit to device 0x52
        Wire.send (0x00); // sends one byte
        Wire.endTransmission (); // stop transmitting
    }
};

#endif

```

Le DS1307 qui est une *Horloge temps réel* commandé par I<sup>2</sup>C.

Adresse I<sup>2</sup>C (7bits) du DS1307 : **1101000**

Figure 4. Data Write—Slave Receiver Mode

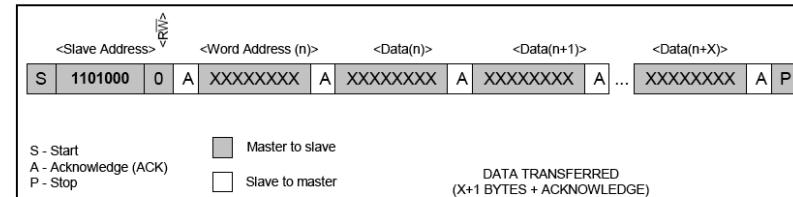
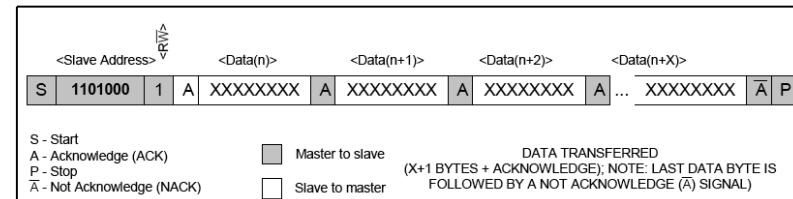


Figure 5. Data Read—Slave Transmitter Mode



Les adresses de registres:

**registre RTC 00h à 07h**

**registre RAM 08h à 3Fh**

Table 2. Timekeeper Registers

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

0 = Always reads back as 0.

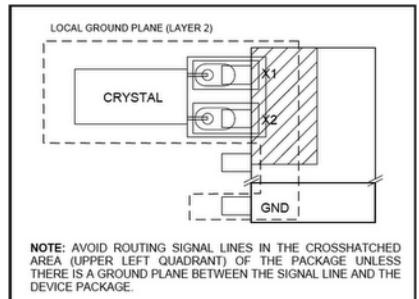
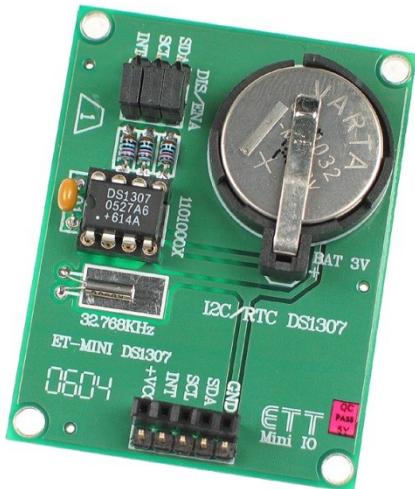
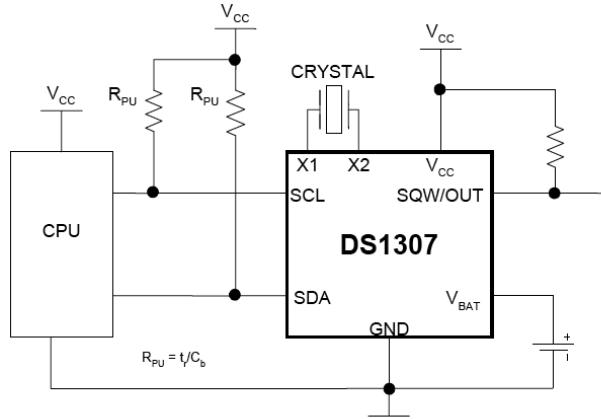
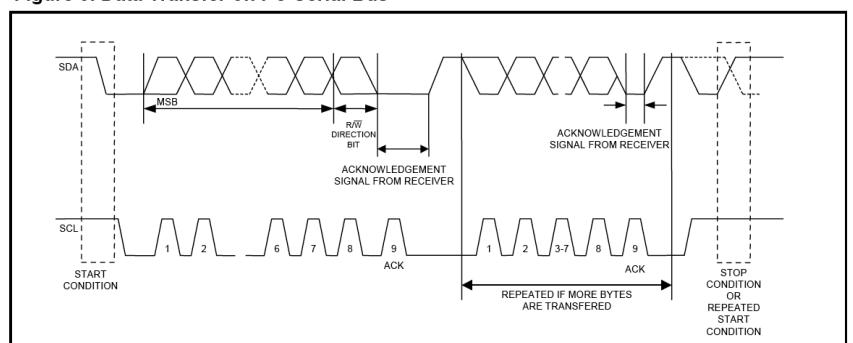


Figure 3. Data Transfer on I<sup>2</sup>C Serial Bus



Chaque bit dans les différents registres ne peuvent être que 0 ou 1.

Comment représenter le contenu du registre en hexadécimal ?

Réponse: On doit d'abord déterminer la représentation binaire et ensuite la convertir en hexadécimal.

Prenons le 3<sup>ème</sup> registre 02H (HOURS) et un exemple d'heure à représenter 12:34 pm :

Lisons de gauche à droite:

- Le Bit 7 est INUTILISE, il est donc à 0.
- Le Bit 6 détermine le format d'affichage de l'heure (12- ou 24-heure): choisissons par exemple 0 pour le format 12-heure
- Le Bit 5 (quand le bit 6 est 0) représente l'indicateur AM/PM : choisissons par exemple 1 pour PM.
- Le Bit 4 représente le bit le plus à gauche de l'heure, i.e. le 1 dans 12:34 pm: choisissons par exemple 1 .
- Les Bits 3 à 0 représentent la conversion binaire du 2 de 12 soit 0010.

Ainsi, le 3<sup>ème</sup> registre permet de représenter 12pm en écrivant le binaire 00110010

Que l'on convertit alors en hexadecimale soit 0x32, à envoyer au registre 02H.

Pour coder les minutes, il faudra procéder de la même façon avec le 2<sup>ème</sup> registre 01H!!!

Table 2. Timekeeper Registers

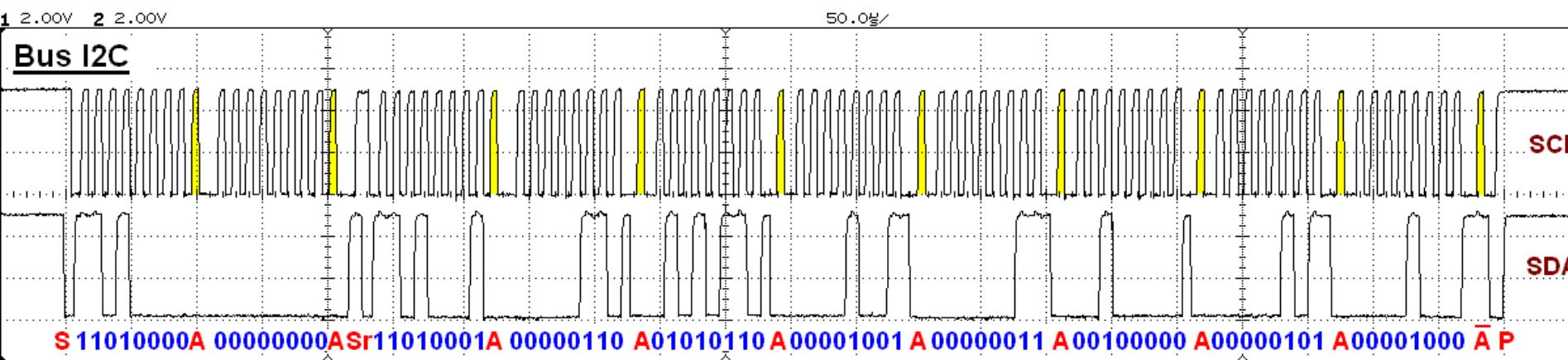
ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00H	CH	10 Seconds			Seconds			Seconds	00–59	
01H	0	10 Minutes			Minutes			Minutes	00–59	
03H	0	0	0	0	0	DAY		Day	01–07	
04H	0	0	10 Date		Date			Date	01–31	
05H	0	0	0	10	Month	Month		Month	01–12	
06H	10 Year				Year			Year	00–99	
07H	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08H-3FH								RAM 56 x 8	00H–FFH	

0 = Always reads back as 0.

Exemple d'utilisation: brancher la fréquence du signal de sortie SQW/OUT (broche 7) à 1 Hz à une des pins d'interruption de l'Arduino

⇒ ce qui provoque une interruption toutes les 1000,000 ms.

⇒ L' Arduino lit l'heure et la date courante dans le DS1307, la communication se fait par le bus I<sup>2</sup>C :



Dans cet exemple de chronogramme, il est 9 heures 56 minutes 6 secondes, le mardi 20 mai 2008.

<http://www.glaicawanderer.com/hobbyrobotics/?p=12>

<http://combustory.com/wiki/index.php/RTC1307 - Real Time Clock>

<http://tronixstuff.wordpress.com/2010/05/20/getting-started-with-arduino-%E2%80%93-chapter-seven/>

<http://tronixstuff.wordpress.com/2010/10/07/add-a-real-time-clock-to-the-freetronics-twentyten/>

## LIAISON SERIE SYNCHRONE: SPI

Le bus I<sup>2</sup>C n'est pas le seul, il est talonné par d'autres circuits série synchrone type SPI ou One Wire. Ces deux dernières appellations sont toutefois moins bien normalisées que le bus I<sup>2</sup>C.

## 1 – SPI «Serial Peripheral Interface»: (ou *Microwire* (pour National Semiconductor) ou bus série 3 fils)

- Standard établi par Motorola et repris par différentes marques
- Bus de données série synchrone, utilisant 4 fils (3 fils + 1 Masse)
- Connexion de type MAITRE-ESCLAVE permettant la connexion de plusieurs circuits => dédié pour établir une communication inter-composants, voir inter-cartes, au sein d'un même système
- Full Duplex (les données voyagent dans les deux directions en même temps) - par opposition à (par exemple), le bus I<sup>2</sup>C (2 fils + masse) qui ne peut pas le faire.
  - Maître-esclaves – Un seul maître possible sur le bus
  - Plusieurs esclaves peuvent coexister sur un bus
  - La sélection du destinataire se fait par une ligne dédiée *chip select (CS)*.

Le bus SPI contient 4 signaux logiques :

- **SCLK** — Horloge (car BUS SERIE SYNCHRONE) (généré par le maître)
- **MOSI** — Master Output, Slave Input (généré par le maître)
- **MISO** — Master Input, Slave Output (généré par l'esclave)
- **SS** — Slave Select, Esclave Actif à l'état BAS (généré par le maître)

Si niveau HAUT le composant ignore ce qui se passe sur le bus => plusieurs composants périphériques qui se partagent les 3 lignes

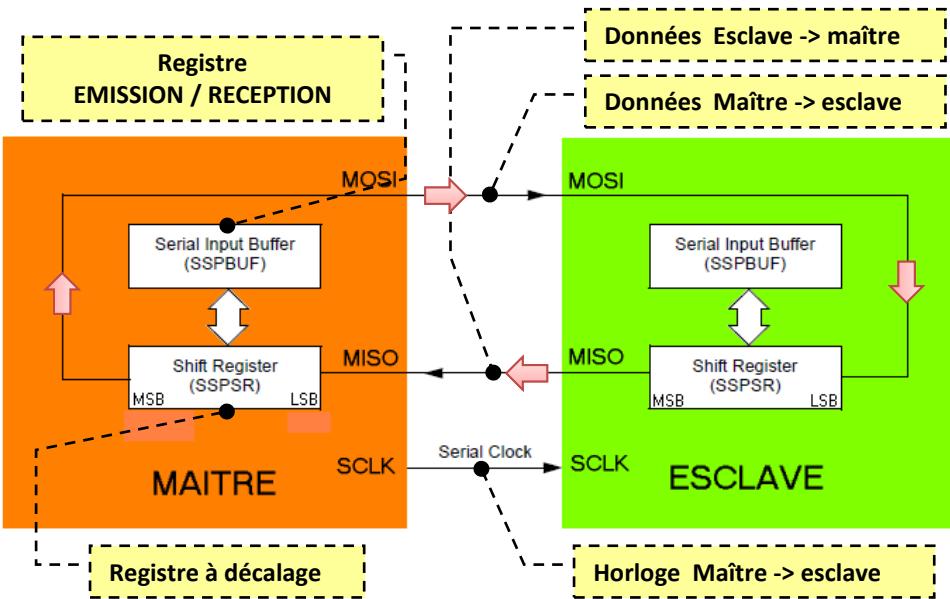


Attention, il existe d'autres noms: **SDI, DI, SI** — Serial Data IN ⇔ MISO

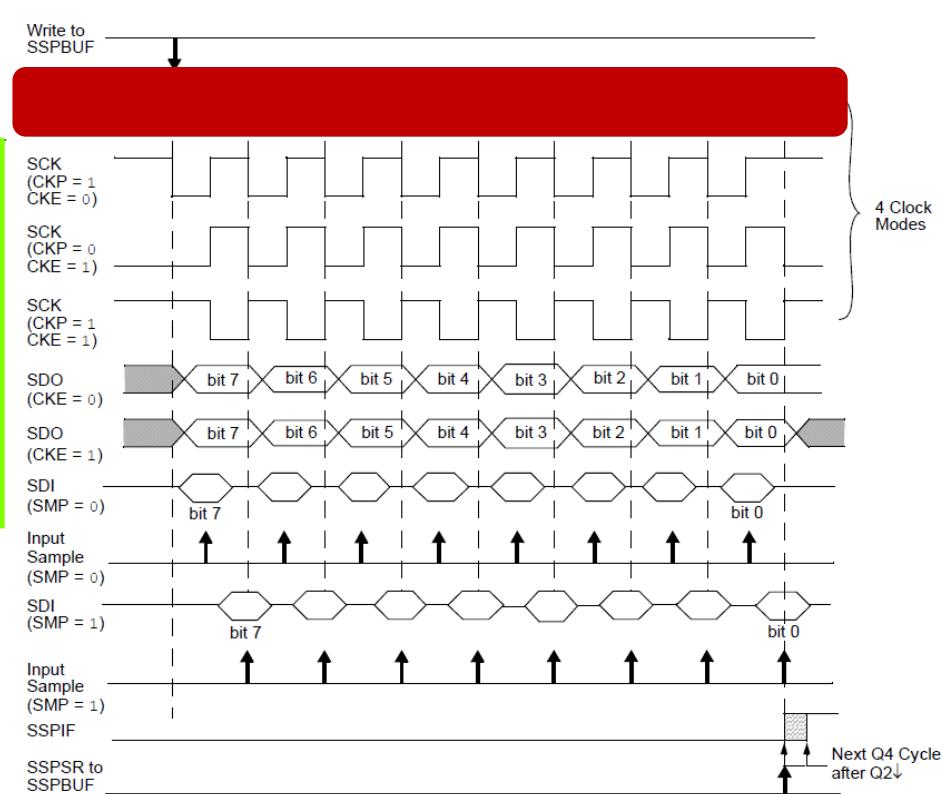
**SDO, DO, SO** — Serial Data OUT ⇔ MOSI

**nCS, CS, nSS, STE** ⇔ SS

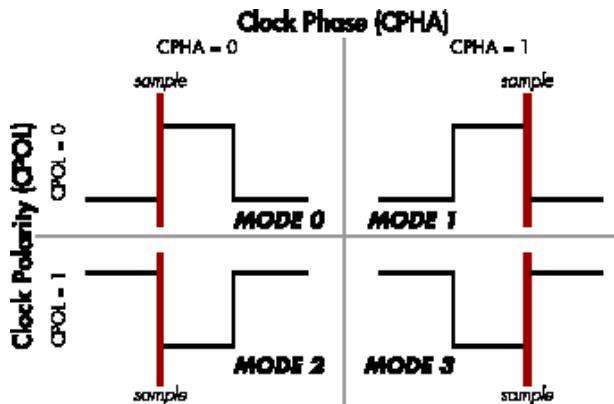
# LE BUS SPI – CHRONOGRAMME GENERAL



- 1 – L'émetteur copie SSPBUF du maître dans le SSPSR
- 2 – Les données de sortie sont envoyées dans SSPSR du récepteur aux coups d'horloge
- 3 – Le récepteur copie SSPSR dans son SSPBUF



CPOL et CPHA ont deux états possibles:  
4 possibilités de configuration.

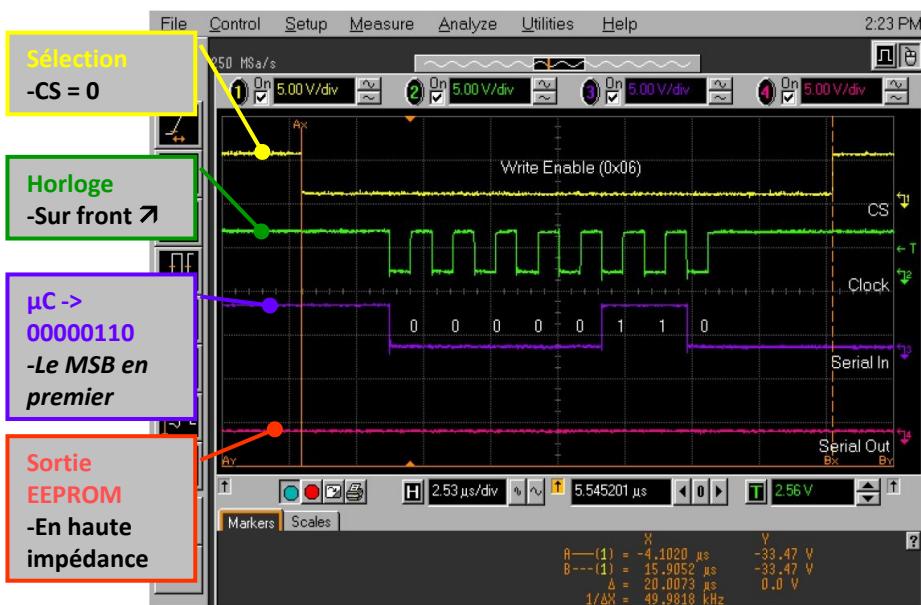


(Maître et esclave doivent avoir les mêmes paramètres)

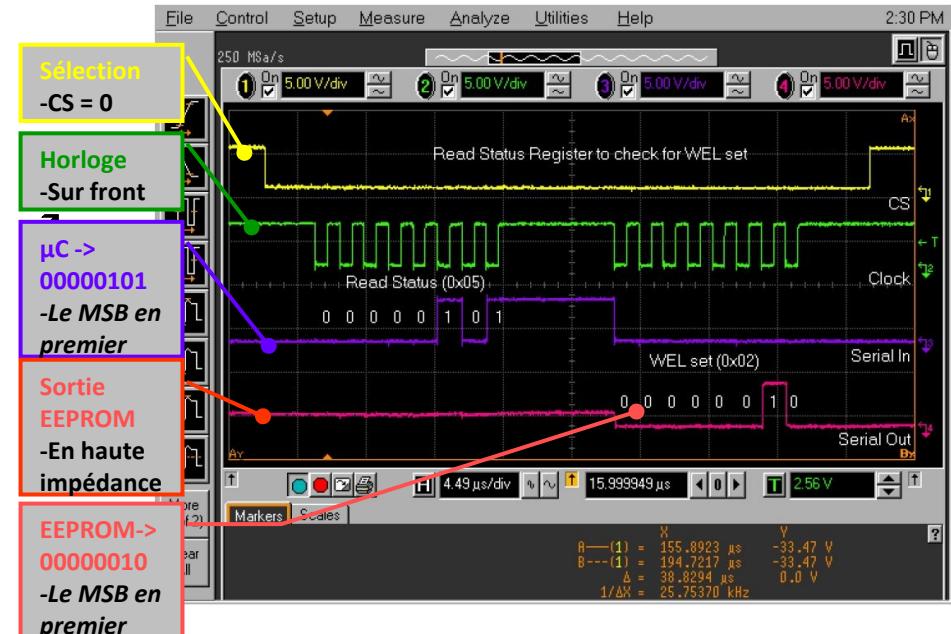
SPI-mode	CPOL (CKP)	CPHA (CKE)
0	0	0
1	0	1
2	1	0
3	1	1

Trois paramètres :

- ♦ La fréquence d'horloge (fixée par le maître).
- ♦ La polarité de l'horloge (paramètre **CPOL** (**Clock polarity**)): détermine le niveau logique de la ligne SCK au repos.
- ♦ La phase de l'horloge (paramètre **CPHA** (**Clock phase**)): détermine le front sur lequel la donnée est modifiée et le front sur lequel la donnée va être lue.



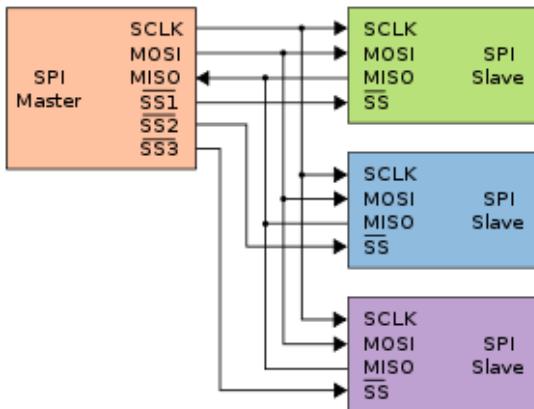
Exemple : autorisation d'écriture dans une EEPROM



Exemple : lecture du registre d'état d'une EEPROM

# LE BUS SPI – CARACTÉRISTIQUES

Plusieurs esclaves mais UN seul esclave actif à la fois :



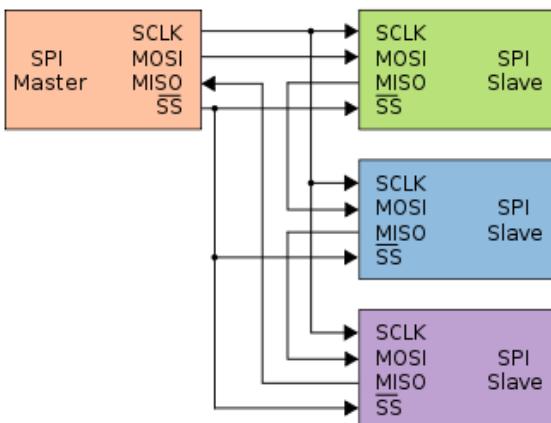
La vitesse :

En mode maître, la vitesse de transmission est sélectionnée par 3 bits du registre SPCON (*Serial Peripheral CONtrol register*): SPR2, SPR1 et SPR0.

La fréquence d'horloge est choisie parmi 7 fréquences obtenues par division de la fréquence de fonctionnement du microcontrôleur.

Plusieurs esclaves (daisy chain): sélection simultanée

Certains composants SPI sont conçus pour être **chaînés**, simplifiant ainsi les connexions entre composants, en réduisant le nombre de lignes SS nécessaires.



SPR2 : SPR1 : SPR0	Fréquence de la SPI	Exemple $F_{transmission}$ (quartz à $F_{\mu c}=20MHz$ )
000	$F_{\mu c}/2$	10
001	$F_{\mu c}/4$	5
010	$F_{\mu c}/8$	2.5
011	$F_{\mu c}/16$	1.25
100	$F_{\mu c}/32$	0.625
101	$F_{\mu c}/64$	0.3125
110	$F_{\mu c}/128$	0.156

-La gestion du bus SPI est facile car l'Arduino contient une bibliothèque baptisée SPI.h

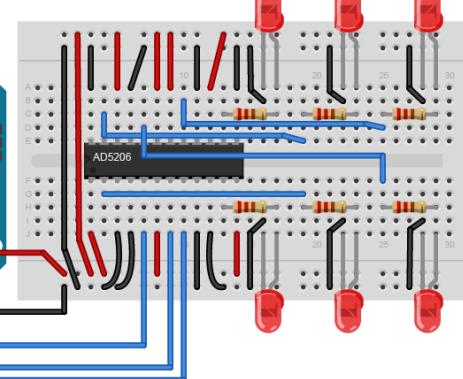
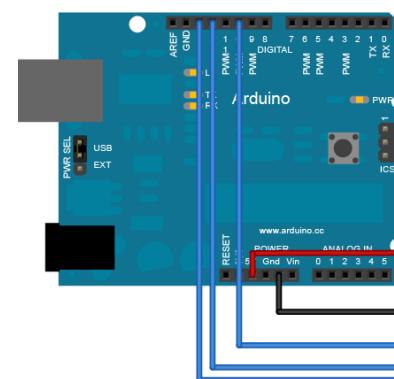
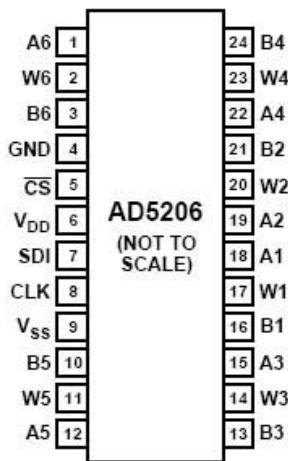
<http://www.arduino.cc/en/Reference/SPI>

<http://tronixstuff.wordpress.com/2011/05/13/tutorial-arduino-and-the-spi-bus/>

## AD5206 Digital Potentiomètre

AD5206 PIN FUNCTION DESCRIPTIONS

Pin No.	Name	Description
1	A6	A Terminal RDAC #6.
2	W6	Wiper RDAC #6, addr = 101 <sub>2</sub> .
3	B6	B Terminal RDAC #6.
4	GND	Ground.
5	CS	Chip Select Input, Active Low. When CS returns high, data in the serial input register is decoded based on the address bits and loaded into the target RDAC latch.
6	V <sub>DD</sub>	Positive power supply, specified for operation at both +3 V or +5 V. (Sum of  V <sub>DD</sub>   +  V <sub>SS</sub>   < 5.5 V.)
7	SDI	Serial Data Input. MSB First.
8	CLK	Serial Clock Input, positive edge triggered.
9	V <sub>SS</sub>	Negative Power Supply, specified for operation at both 0 V or -2.7 V. (Sum of  V <sub>DD</sub>   +  V <sub>SS</sub>   < 5.5 V.)
10	B5	B Terminal RDAC #5.
11	W5	Wiper RDAC #5, addr = 100 <sub>2</sub> .
12	A5	A Terminal RDAC #5.
13	B3	B Terminal RDAC #3.
14	W3	Wiper RDAC #3, addr = 010 <sub>2</sub> .
15	A3	A Terminal RDAC #3.
16	B1	B Terminal RDAC #1.
17	W1	Wiper RDAC #1, addr = 000 <sub>2</sub> .
18	A1	A Terminal RDAC #1.
19	A2	A Terminal RDAC #2.
20	W2	Wiper RDAC #2, addr = 001 <sub>2</sub> .
21	B2	B Terminal RDAC #2.
22	A4	A Terminal RDAC #4.
23	W4	Wiper RDAC #4, addr = 011 <sub>2</sub> .
24	B4	B Terminal RDAC #4.



<http://arduino.cc/en/Tutorial/SPIDigitalPot>

Functions  
[begin\(\)](#)  
[end\(\)](#)  
[setBitOrder\(\)](#)  
[setClockDivider\(\)](#)  
[setDataMode\(\)](#)  
[transfer\(\)](#)

## /\* Digital Pot Control

This example controls an Analog Devices AD5206 digital potentiometer.  
The AD5206 has 6 potentiometer channels. Each channel's pins are labeled

A - connect this to voltage

W - this is the pot's wiper, which changes when you set it

B - connect this to ground.

The AD5206 is SPI-compatible, and to command it, you send two bytes, one with the channel number (0 - 5) and one with the resistance value for the channel (0 - 255).

The circuit:

- \* All A pins of AD5206 connected to +5V
- \* All B pins of AD5206 connected to ground
- \* An LED and a 220-ohm resistor in series connected from each W pin to ground
- \* CS - to digital pin 10 (SS pin)
- \* SDI - to digital pin 11 (MOSI pin)
- \* CLK - to digital pin 13 (SCK pin)

created 10 Aug 2010 by Tom Igoe

*Thanks to Heather Dewey-Hagborg for the original tutorial, 2005*

\*/

```

// include the SPI library:
#include <SPI.h>
// set pin 10 as the slave select for the digital pot:
const int slaveSelectPin = 10;

void setup() {
  // set the slaveSelectPin as an output:
  pinMode (slaveSelectPin, OUTPUT);
  // initialize SPI:
  SPI.begin();
}

void loop() {
  // go through the six channels of the digital pot:
  for (int channel = 0; channel < 6; channel++) {
    // change the resistance on this channel from min to max:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, level);
      delay(10);
    }
    // wait a second at the top:
    delay(100);
    // change the resistance on this channel from max to min:
    for (int level = 0; level < 255; level++) {
      digitalPotWrite(channel, 255 - level);
      delay(10);
    }
  }
}

int digitalPotWrite(int address, int value) {
  // take the SS pin low to select the chip:
  digitalWrite(slaveSelectPin,LOW);
  // send in the address and value via SPI:
  SPI.transfer(address);
  SPI.transfer(value);
  // take the SS pin high to de-select the chip:
  digitalWrite(slaveSelectPin,HIGH);
}

```

<http://arduino.cc/en/Tutorial/SPIDigitalPot>

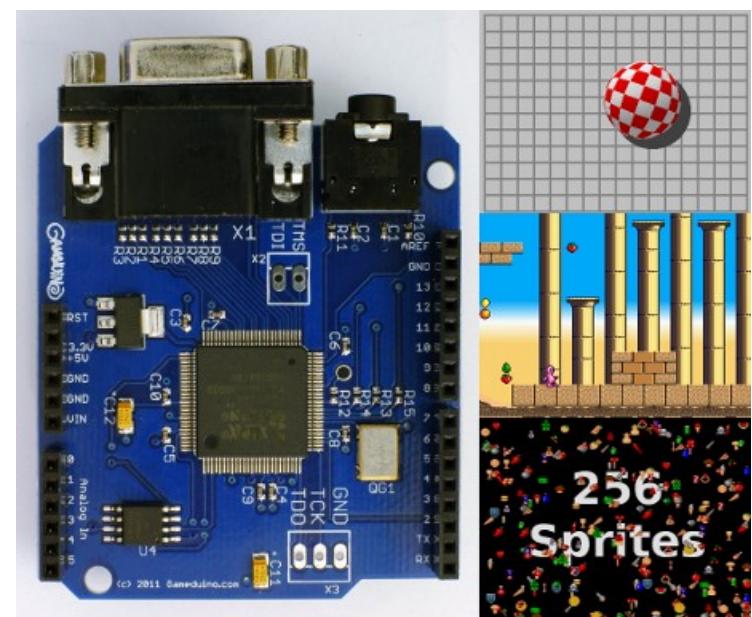
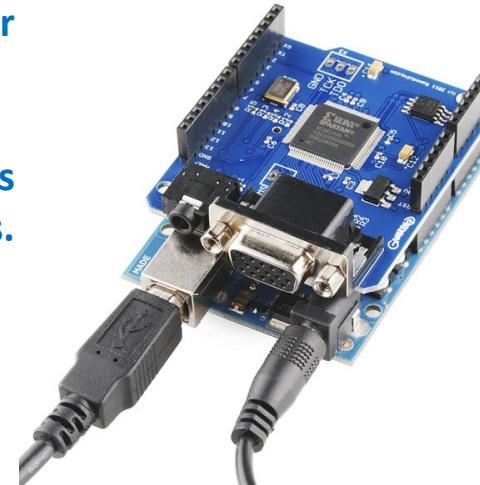
Gameduino est un shield Arduino disposant d'un contrôleur graphique et sonore FPGA de type Xilinx.

Son contrôle se fait par le bus SPI .

Le code sert à envoyer les commandes à la carte pour gérer les registres à bascule, la mémoire et les fonctions graphiques.

Les données sont alors gérées par le FPGA Xilinx qui permet :

- video output is 400x300 pixels in 512 colors
- all color processed internally at 15-bit precision
- compatible with any standard VGA monitor (800x600 @ 72Hz)
- background graphics
  - 512x512 pixel character background
  - 256 characters, each with independent 4 color palette
  - pixel-smooth X-Y wraparound scroll
- foreground graphics
  - each sprite is 16x16 pixels with per-pixel transparency
  - each sprite can use 256, 16 or 4 colors
  - four-way rotate and flip
  - pixel-perfect sprite collision detection
- audio output is a stereo 12-bit frequency synthesizer
- 64 independent voices 10-8000 Hz
- per-voice sine wave or white noise
- sample playback channel



<http://excamera.com/sphinx/gameduino/>

# APPLICATION DU SPI: VERS LE FPGA

Application Note: Spartan-3A FPGAs



XAPP974 (v1.1.3) March 24, 2009

## Indirect Programming of SPI Serial Flash PROMs with Spartan-3A FPGAs

Author: Jameel Hussein

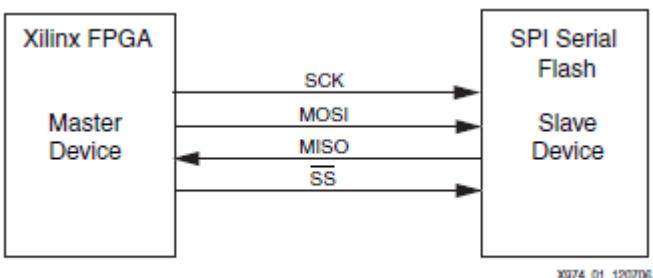
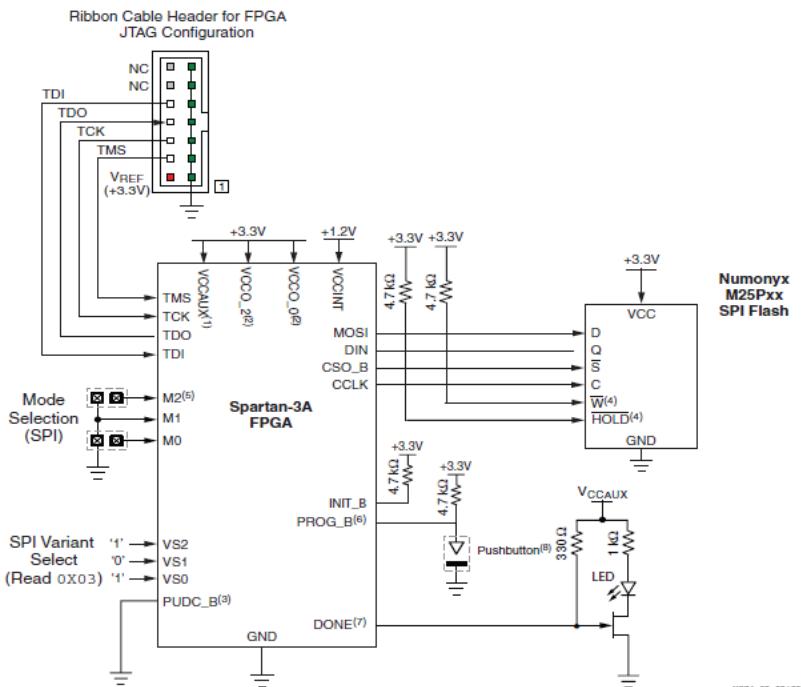


Figure 1: Basic Block Diagram for SPI Configuration Mode



[http://www.xilinx.com/support/documentation/application\\_notes/xapp974.pdf](http://www.xilinx.com/support/documentation/application_notes/xapp974.pdf)

# APPLICATION DU SPI: VERS LE FPGA

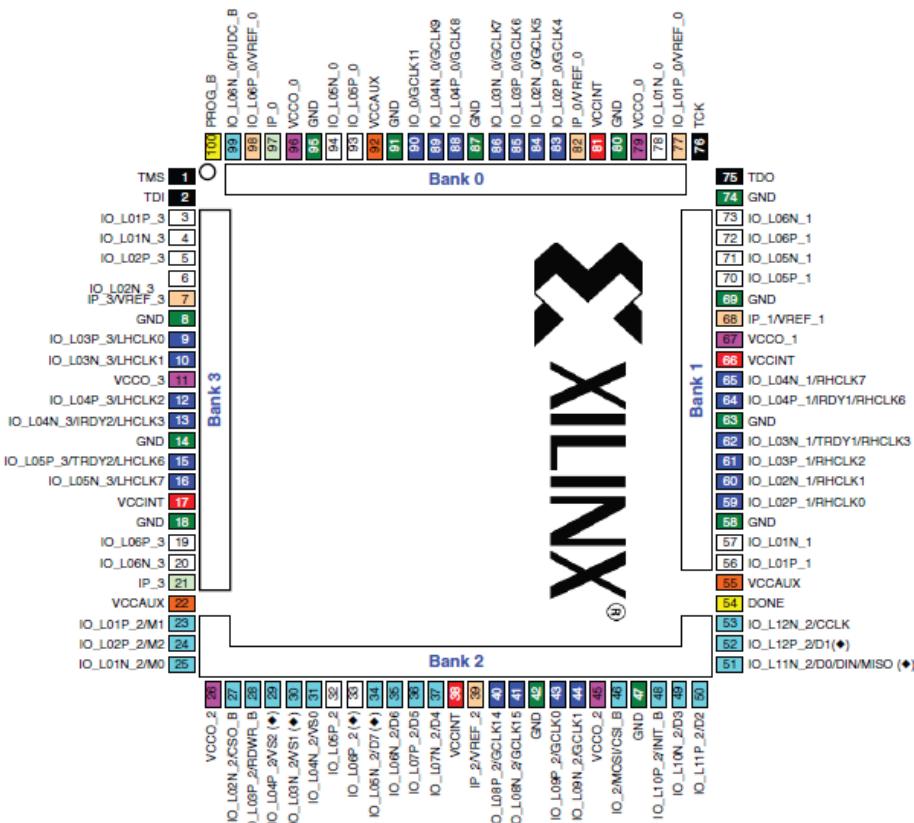


Figure 17: VQ100 Package Footprint - XC3S50A (Top View)

17	I/O: Unrestricted, general-purpose user I/O	20	DUAL: Configuration pins, then possible user I/O	6	VREF: User I/O or input voltage reference for bank
2	INPUT: Unrestricted, general-purpose input pin	23	CLK: User I/O, input, or global buffer input	6	VCCO: Output voltage supply for bank
2	CONFIG: Dedicated configuration pins	4	JTAG: Dedicated JTAG port pins	4	VCCINT: Internal core supply voltage (+1.2V)
0	N.C.: Not connected	13	GND: Ground	3	VCCAUX: Auxiliary supply voltage

[http://www.xilinx.com/support/documentation/data\\_sheets/ds529.pdf](http://www.xilinx.com/support/documentation/data_sheets/ds529.pdf)

# GAMEDUINO REFERENCE

## screen

Screen RAM is a grid of  $64 \times 64$  bytes, each byte is a single character index. This byte controls the character image and palette used for that  $8 \times 8$  pixel cell. The total size of the background screen is  $512 \times 512$  pixels, but only  $400 \times 300$  pixels are visible. The **SCROLL\_X** and **SCROLL\_Y** registers control the position of this  $400 \times 300$  pixel window within the larger screen area.



## characters

Characters are  $8 \times 8$  grids of pixels, defined by the values in the character data and palette RAMs. The character data RAM holds the 64 pixels of the character image, encoded using two bits per pixel. The hardware uses these two bits to look up the final color in the character's 4-entry palette.

For example, a character with a palette of blue, yellow, red and white might appear as shown below. In the left-hand square, the pixel values 0–3 are shown. In the middle square, these pixel values in binary are listed. In the right hand column are the hex values, as they appear in memory for this character.

0	1	1	1	1	1	1	1	00 01 01 01	01 01 01 01	15	55
0	1	2	2	2	2	1	1	00 01 10 10	10 01 01 01	1A	95
1	1	2	2	2	2	1	1	01 01 10 10	10 01 01 00	5A	94
0	1	2	3	3	3	3	3	00 01 10 11	11 11 11 11	1B	FF
1	1	1	1	1	2	1	1	01 01 01 01	10 01 01 01	55	95
0	1	1	1	1	2	1	1	00 01 01 01	10 01 01 01	15	95
1	1	1	1	1	2	1	1	01 01 01 01	10 01 01 01	55	95
0	1	1	1	1	2	1	1	00 01 01 01	10 01 01 01	15	95

## memory map

Gameduino has 32 kbytes of memory, organized into different functions. The background section controls background character graphics. The sprite section controls the foreground sprite graphics.

background	0x0000	64x64 character screen
	0x0FFF	character data 256 characters of $8 \times 8$ pixels
	0x2000	character palettes 256 characters of 4 colors
control registers	0x27FF	control registers collision RAM
	0x2800	
	0x2FFF	sprite control 256x4 bytes
sprites	0x3000	
	0x37FF	sprite palette 4 palettes of 256 colors
	0x3800	
	0x3FFF	sprite images 64 images of $16 \times 16$ bytes
	0x2900	
	0x7FFF	

## registers

Registers control some simple functions of the Gameduino. Gameduino is little-endian, so 16-bit registers have their lower 8 bits at the lower address in memory.

address	bytes	name	description	access	reset value
0x2800	1	IDENT	Gameduino identification—always reads as 0x8D	r	0x8D
0x2801	1	REV	Hardware revision number. High 4 bits are major revision, low 4 bits are minor	r	0x10
0x2802	1	FRAME	Frame counter, increments at the end of each displayed frame	r	0
0x2803	1	VBLANK	Set to 1 during the video blanking period	r	0
0x2804	2	SCROLL_X	Horizontal background scrolling register, 0–511	r/w	0
0x2806	2	SCROLL_Y	Vertical background scrolling register, 0–511	r/w	0
0x2808	1	JK_MODE	Sprite collision class mode enable 0–1	r/w	0
0x280A	1	SPR_DISABLE	Sprite control: 0=enable sprite display, 1=disable sprite display	r/w	0
0x280B	1	SPR_PAGE	Sprite page select: 0=display from locations 0x3000–0x33FF, 1=from 0x3400–0x37FF	r/w	0
0x280C	1	IMODE	Pin 2 mode: 0=disconnect, 0x48=flash enable, 0x4A=coprocessor control	r/w	0
0x280E	2	BG_COLOR	Background color	r/w	0
0x2810	2	SAMPLE_L	Audio left sample value, 16 bit signed -32768 to +32767	r/w	0
0x2812	2	SAMPLE_R	Audio right sample value, 16 bit signed -32768 to +32767	r/w	0
0x281E	2	SCREENSHOT_Y	Screenshot line select 0–299	r/w	0
0x2840	32	PALETTE 16A	16-color sprite A palette	r/w	0000 (black)
0x2860	32	PALETTE 16B	16-color sprite B palette	r/w	0000 (black)
0x2880	8	PALETTE 4A	4-color sprite A palette	r/w	0000 (black)
0x28B8	8	PALETTE 4B	4-color sprite B palette	r/w	0000 (black)
0x2900	256	COLLISION	Collision RAM	r	0
0x2A00	256	VOICES	Audio voice controls	r/w	0
0x2B00	800	SCREENSHOT	Screenshot line RAM	r	0



# SPRITES



**Au début de jeux vidéos, les sprites « hardware » étaient une méthode permettant de gérer des bitmaps semblant faire partie d'une seule image sur un écran.**

Computer, chip	Year	Sprites on screen	Sprites on line	Max. texels on line	Texture width	Texture height	Colors	Hardware zoom	Rotation	Background	Collision detection	Transparency	Source
Amiga, Denise	1985	Display list	8	?	16	Arbitrary	3, 15	Vertical by display list	No	2 bitmap layers	Yes	Color key	
Amiga (AGA), Lisa	1992	Display list	8	?	16, 32, 64	Arbitrary	3, 15	Vertical by display list	No	2 bitmap layers	Yes	Color key	
Amstrad Plus, Asic	1990	Display list run by CPU	16 min.	?	16	16	15	1, 2, 4x vertical, 1, 2, 4x horizontal	No	Bitmap layer	No	Color key	[7]
Atari 2600, TIA	1977	Multipled by CPU	9 (with triplication)	51 (with triplication)	1, 8	262	1	1, 2, 4, 8x horizontal	Horizontal mirroring	1 bitmap layer	Yes	Color key	[8]
Atari 8-bit, GTIA/ANTIC	1979	Display list	8	40	2, 8	128, 256	1,3	1, 2x vertical, 1, 2, 4x horizontal	No	1 tile or bitmap layer	Yes	Color key	[9]
C64, VIC-II	1982	Display list run by CPU	8	96, 192	12, 24	21	1, 3	1, 2x integer	No	1 tile or bitmap layer	Yes	Color key	[10]
Game Boy	1989	40	10	80	8	8, 16	3	No	No	1 tile layer	No	Color key	[11]
GBA	2001	128	128	1210	8, 16, 32, 64	8, 16, 32, 64	15, 255	Yes affine	Yes affine	4 layers, 2 layers, and 1 affine layer, 2 affine layers	No	Color key, blending	[12]
Gameduino	2011	256	96	1,536	16	16	255	No	Yes	1 tile layer	Yes	Color key	[13]
NES, RP2C0x	1983	64	8	64	8	8, 16	3	No	Horizontal and vertical mirroring	1 tile layer	Partial	Color key	[15]
Neo Geo	1990	384	96	1536	16	Up to 512	15	sprite shrinking	No	No	No	Color key	.
Out Run, dedicated hardware	1986	128	32	?	8	8	?	Yes anisotropic	No	3 tile layers	?	Alpha	[16][17]
PC Engine, HuC6270A	1987	64	16	256	16, 32	16, 32, 64	15	No	No	1 tile layer	Yes	Color key	
Sega Master System Sega Game Gear	1985	64	8	64	8	8, 16	15	1, 2x integer	No	1 tile layer	Yes	Color key	[18]
Mega Drive	1988	80	20	320	8, 16, 24, 32	8, 16, 24, 32	15	No	No	2 tile layers	Yes	Color key	[19]
Sharp X68000	1987	128	32	?	16	16	15	No	No	?	?	Color key	
SNES	1990	128	34	272	8, 16, 32, 64	8, 16, 32, 64	15	Background only	Background only	3 tile layers or 1 affine mapped tile layer	Yes	Color key, averaging	
Texas Instruments TMS9918	1979	32	4	64	8, 16	8, 16	1	1, 2x integer	No	1 tile layer	Partial	Color key	[20]
Yamaha V9938	1986	32	8	128	8, 16	8,16	1, 3, 7, 15 per line	1, 2x integer	No	1 tile or bitmap layer	Partial	Color key	
Yamaha V9958	1988	32	8	128	8, 16	8,16	1, 3, 7, 15 per line	1, 2x integer	No	1 tile or bitmap layer	Partial	Color key	
Computer, chip	Year	Sprites on screen	Sprites on line	Max. texels on line	Texture width	Texture height	Colors	Hardware zoom	Rotation	Background	Collision detection	Transparency	Source

[http://en.wikipedia.org/wiki/Sprite\\_%28computer\\_graphics%29#Hardware\\_sprites](http://en.wikipedia.org/wiki/Sprite_%28computer_graphics%29#Hardware_sprites)



### Inconvénients :

- Monopolise plus de pattes d'un boîtier que l'I<sup>2</sup>C ou une UART qui en utilisent seulement deux.
- Aucun adressage possible, il faut une ligne de sélection par esclave en mode non chaîné.
- Le protocole n'a pas d'acquittement. Le maître peut parler dans le vide sans le savoir.
- Il ne peut y avoir qu'un seul maître sur le bus.
- Ne s'utilise que sur de courtes distances contrairement aux protocoles RS-232, RS-485 ou bus CAN



### Avantages :

- Communication Full duplex
- Débit assez important par rapport à I<sup>2</sup>C
- Flexibilité du nombre de bits à transmettre
- Simplicité de l'interface matérielle
- Aucun arbitre nécessaire car aucune collision possible
- Les esclaves utilisent l'horloge du maître et n'ont donc pas besoin d'oscillateur de précision
- Partage d'un bus commun pour l'horloge, MISO et MOSI entre les périphériques
- Les esclaves n'ont pas besoin d'un ID unique — contrairement à l'I<sup>2</sup>C, au GPIB et au SCSI

## 2 – Un fil ou ONE WIRE créé par Dallas (société aujourd’hui appelée MAXIM)

- La technologie "One Wire" est un bus de communication série permettant l'adressage et l'utilisation de plusieurs capteurs à communication série sur une seule et même broche numérique.

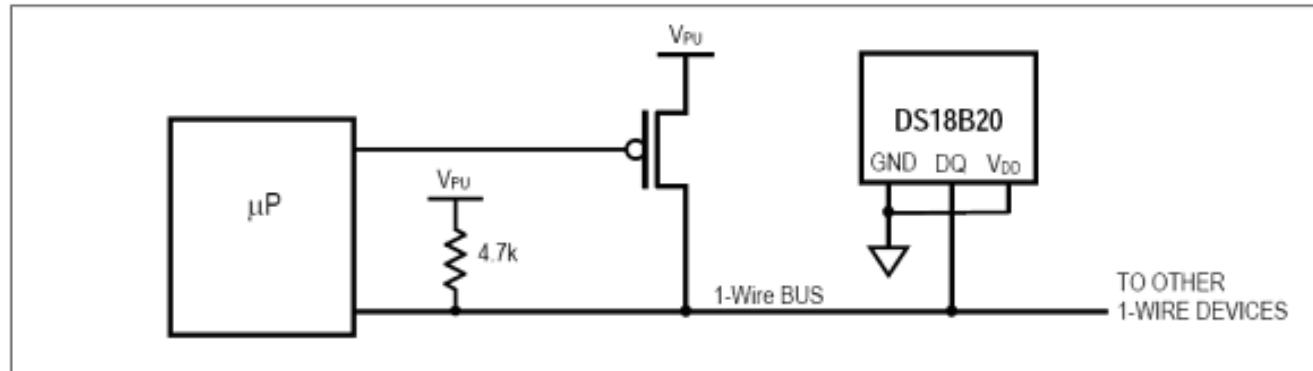
- Bus de données série synchrone n'utilisant qu'un seul fil (+ 1 masse de référence de potentiel) .

- Le fil permet de véhiculer les données dans les deux sens, chaque composant pouvant être émetteur et récepteur à tout instant.

- Mais il permet aussi de les alimenter (mode d'alimentation parasite) pour les moins gourmands !

- La gestion du bus ONE WIRE est facile car l'Arduino contient une bibliothèque tierce baptisée: **OneWire.h** (auteur: Jim Studt)

Figure 4. Supplying the Parasite-Powered DS18B20 During Temperature Conversions



[http://www.milesburton.com/?title=Dallas\\_Temperature\\_Control\\_Library](http://www.milesburton.com/?title=Dallas_Temperature_Control_Library)

<http://www.arduino.cc/playground/Learning/OneWire>

**I<sup>2</sup>C****Vitesse de transmission****SPI**

Même s'il existe des variations de l'I<sup>2</sup>C qui montent au dessus de 1MHz, la grande majorité des implémentations que l'on trouve utilisent généralement **100 ou 400 kHz**

Pour le SPI il est possible de trouver certains composants au delà de **20 Mbits**

**I<sup>2</sup>C****Topologie****SPI**

C'est un véritable protocole qui permet l'interconnexion de multiples boîtiers dans différentes configurations : Maitre /Esclave, Maitre/Multiple esclaves, Multiple Maitres/Multiples esclaves

En général point à point, bien que l'on puisse connecter plusieurs esclaves mais il faut alors des lignes supplémentaires. **Un seul maître qui génère l'horloge.**

**I<sup>2</sup>C****Consommation****SPI**

du à la configuration collecteur/ drain ouvert sur les 2 lignes de transmission (SDA + SCL), **consommation relativement élevée**

signaux de type TTL/CMOS donc **consommation faible**

**Avantages/Inconvénients**

- ✓ Si l'on doit interconnecter plusieurs boîtiers et que la vitesse n'est pas un problème, préférer l'I<sup>2</sup>C car c'est un protocole (*ce qui n'est pas le cas du SPI*)
- ✓ Si on veut de la vitesse le SPI est loin devant...
- ✓ Implémentation logicielle sur des E/S : il est BEAUCOUP plus facile (et cela prend moins de ressources) de faire du SPI par logiciel sur des broches d'E/S que de l'I<sup>2</sup>C dû à la machine d'état.
- ✓ Mise en œuvre : l'I<sup>2</sup>C est plus compliqué à mettre en œuvre (il suffit de voir le nombre de questions sur l'I<sup>2</sup>C dans les forums...).
- ✓ L'interconnexion de plusieurs boîtiers est également plus délicate avec l'I<sup>2</sup>C car il faut prendre en compte les impédances de chacun des boîtiers pour calculer les résistances de rappels.

**Bus One Wire :**

**La technologie One Wire est une technologie comparable à la technologie I<sup>2</sup>C (en mieux ?).**

### Bus I<sup>2</sup>C - Inter Integrated Circuit:

Principalement développé pour la Domotique et l'électronique domestique, ce protocole est très populaire et donc utilisé par de nombreux composants.

Le protocole I<sup>2</sup>C ne requiert que 3 fils pour fonctionner et peut être implémenté sur n'importe quel microcontrôleur.

Supporte plusieurs maîtres (donc risque de collision)

### Bus SPI - Serial Peripheral Interface:

Le bus SPI est Full Duplex et basé sur une communication maître-esclave.

Le bus SPI supporte plusieurs esclaves mais un seul peut communiquer à la fois avec le maître (donc logique de sélection!)

### Bus One Wire :

La technologie One Wire est une technologie comparable à la technologie I<sup>2</sup>C (en mieux ?).

# EXEMPLE D'APPLICATION DE LA LIAISON SERIE



to Wi-Fi

to Ethernet



to graphic LCD

to 8-servo controller



to Roomba

Lantronix Wi-Port and  
Lantronix Xport  
<http://lantronix.com/>

Seetron Serial Graphic display and  
Mini SSC  
<http://www.seetron.com/slcds.htm>  
<http://www.seetron.com/ssc.htm>

“Hacking Roomba”,  
out in a few weeks,  
<http://hackingroomba.com/>

## III – 8 LIBRAIRIES

## III – 8-1 EXEMPLE DE LIBRAIRIE

L'ATMEGA des cartes Arduino possède une EEPROM: mémoire dont les valeurs sont conservées lorsque la carte est éteinte:

- 512 octets sur l'ATmega168,
- 1024 octets (1 Ko) sur l'ATmega328,
- 4096 octets (4 Ko) sur l'ATmega8,

La bibliothèque EEPROM vous permet de lire et d'écrire ces octets:

#### Exemple de lecture Syntax

```
#include <EEPROM.h>
int a = 0;
int value;
```

```
void setup()
{
  Serial.begin(9600);
}
```

```
void loop()
{
  value = EEPROM.read(a);
  Serial.print(a);Serial.print("\t");
  Serial.print(value);
  Serial.println();
  a = a + 1;
  if (a == 512)
    a = 0;
  delay(500);
}
```

#### Exemple d'écriture Syntax

```
EEPROM.write(address, value)
```

```
#include <EEPROM.h>

void setup()
{
  for (int i = 0; i < 512; i++)
    EEPROM.write(i, i);
}
void loop() {
```

```
/* EEPROM Clear: Sets all of the bytes of the EEPROM to 0.*/
#include <EEPROM.h>
void setup()
{
  // write a 0 to all 512 bytes of the EEPROM
  for (int i = 0; i < 512; i++) EEPROM.write(i, 0);
}
void loop() {
```

*Note: une écriture EEPROM prend 3,3 ms. La mémoire EEPROM a une durée de vie spécifiée de 100.000 cycles d'écriture/effacement. Soyez prudent dans la façon de l'utiliser.*

## III – 8 - 2 CREER UNE LIBRAIRIE

Lorsque vous commencez à faire de gros projets, il devient utile voire indispensable de (très) bien organiser son code:  
Séparez votre code en différents fichiers afin d'avoir des entités logiques séparées les unes des autres.

Généralement, on fait un fichier par unités “logiques”.

*En pratique: pour créer un nouveau fichier dans l'IDE Arduino, il suffit de cliquer sur la petite flèche en haut de l'espace d'édition du code puis ensuite de cliquer sur “Nouvel Onglet” ou “New Tab” comme mis en évidence sur la capture d'écran ci-dessous :*

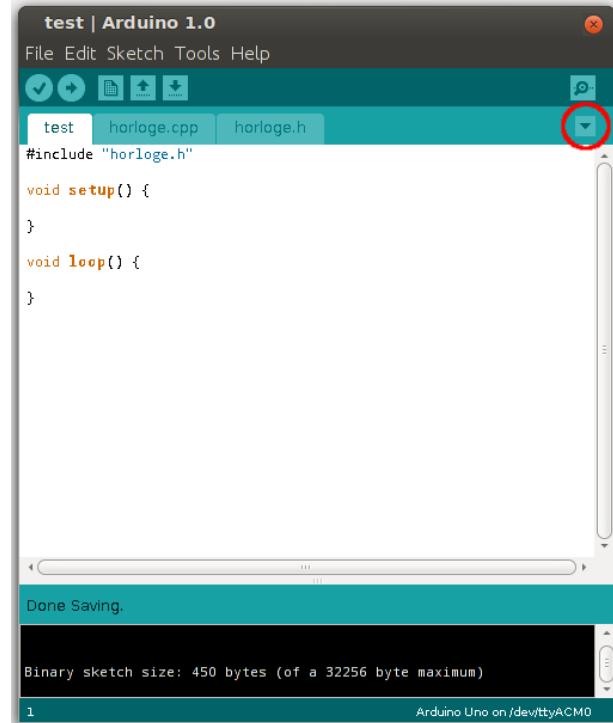
Attention, à chaque fois que l'on veut créer un **nouveau fichier de code** on ne va pas en créer un mais deux !

**1 - Le fichier .h:** Le premier fichier aura l'extension .h signifiant **header**.

Ce fichier va regrouper les **prototypes des fonctions** ainsi que les définitions de structures ou de classes mais nous verrons cela après.

**Prototype d'une fonction** : définit le nom de la fonction, ce qui rentre à l'intérieur (les paramètres) et ce qui en sort (la variable de retour).

Ainsi, votre programme principal aura une idée de comment fonctionne *extérieurement* votre fonction.



```
test | Arduino 1.0
File Edit Sketch Tools Help
[tab icons] test horloge.cpp horloge.h
#include "horloge.h"

void setup() {
}

void loop() {
}

Done Saving.

Binary sketch size: 450 bytes (of a 32256 byte maximum)
1 Arduino Uno on /dev/ttyACM0
```

**Exemple:** Imaginons un fichier de gestion d'horloge RTC: **horloge.h**

```
char getHeure(); char getMinute(); char getSeconde();
char getJour(); char getMois();
char getAnnee(); void setHeure(char val);
void setMinute(char val); void setSeconde(char val);
void setJour(char val); void setMois(char val);
void setAnnee(char val); void afficherDate();
void afficherHeure(); void afficherDateHeure();
```

Comme vous pouvez le voir, avec ces définitions on peut savoir ce qu'est supposé faire la fonction grâce à son nom et le type de variable qu'elle manipule en entrée et en sortie.

## 2 - Le second fichier .cpp:

Il sert à implémenter les fonctions définies dans le .h,

i.e écrire le contenu de vos fonctions,

Première étape: inclure le fichier de prototypes via la commande de préprocesseur #include :

Par exemple:

```
#include "horloge.h" // horloge.h pour notre exemple
```

Attention: cette ligne doit être la **première** de votre fichier .cpp et elle ne prend pas de ; à la fin.

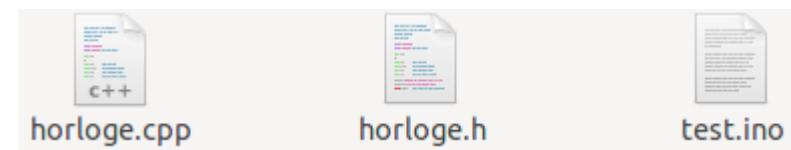
## 3 - Lier nos fichiers au programme principal:

1 - il faut s'assurer que vos fichiers .h et .cpp sont dans le même dossier que votre .ino

2 - Inclure le fichier .h en haut de votre fichier: #include "horloge.h"

## Codes des fonctions :

```
/* fichier horloge.cpp */
#include "horloge.h"
char getHeure() { Serial.println("getHeure"); return 0; }
char getMinute() { Serial.println("getMinute"); return 0; }
char getSeconde() { Serial.println("getSeconde"); return 0; }
char getJour() { Serial.println("getJour"); return 0; }
char getMois() { Serial.println("getMois"); return 0; }
char getAnnee() { Serial.println("getAnnee"); return 0; }
void setHeure(char val) { Serial.print("setHeure : "); Serial.println(val, DEC); }
void setMinute(char val) { Serial.print("setMinute : "); Serial.println(val, DEC); }
void setSeconde(char val) { Serial.print("setSeconde : "); Serial.println(val, DEC); }
void setJour(char val) { Serial.print("setJour : "); Serial.println(val, DEC); }
void setMois(char val) { Serial.print("setMois : "); Serial.println(val, DEC); }
void setAnnee(char val) { Serial.print("setAnnee : "); Serial.println(val, DEC); }
void afficherDate() { Serial.println("afficherDate"); }
void afficherHeure() { Serial.println("afficherHeure"); }
void afficherDateHeure() { Serial.println("afficherDateHeure"); }
```



## 4 - Utilisation:

Vous pouvez maintenant faire des appels tout simples à vos fonctions personnalisées dans le programme (setup(), loop(), des fonctions...!).

Maintenant, quand vous allez compiler, le compilateur va aller chercher le fichier pointé par le include, le compiler puis le lier dans votre programme principal.

Remarque1 : il peut arriver que le lien avec les symboles/librairies Arduino ne se fasse pas correctement. Dans ce cas là, rajoutez l'include suivant au début de votre .h ou .cpp : #include "Arduino.h"

Remarque2: avec cette technique: vous pouvez coder en C++ pour créer des classes et ainsi pousser l'organisation encore plus loin !

- La création de bibliothèques personnalisées permet de simplifier la réutilisation de code.
- Beaucoup de bibliothèques sont déjà présentes et intégrés au logiciel Arduino.
- Elles vous permettent de programmer plus vite et plus simplement vos systèmes.

Nous allons voir comment créer une bibliothèque à travers un exemple concret : la bibliothèque Morse.

*Exemple tiré du tutoriel officiel de création de bibliothèque:  
<http://arduino.cc/en/Hacking/LibraryTutorial>*

**Attention: Library est un faux ami. La traduction française est Bibliothèque et non pas Librairie.**

Examinons le code ci-contre:

les parties de ce code que nous mettrons dans notre nouvelle bibliothèque sont clairement `pinMode(pin, OUTPUT)` et les fonctions `dot()` et `dash()`.

Voyons maintenant comment créer une bibliothèque, e.g. Morse.

### Code C

```
int pin = 13;

void setup()
{
    pinMode(pin, OUTPUT);
}

void loop()
{
    // SOS
    // 3x POINT 3x TIRET 3xPOINT
    dot(); dot(); dot();
    dash(); dash(); dash();
    dot(); dot(); dot();
    delay(3000);
}

void dot() // point
{
    digitalWrite(pin, HIGH);
    delay(250);
    digitalWrite(pin, LOW);
    delay(250);
}

void dash() // tiret
{
    digitalWrite(pin, HIGH);
    delay(1000);
    digitalWrite(pin, LOW);
    delay(250);
}
```

**STRUCTURE D'UNE BIBLIOTHÈQUE:** Les bibliothèques Arduino sont composées d'au moins **2 fichiers**:

- un fichier d'en-tête finissant par **.h** (qui contient les définitions des fonctions disponibles) et
- un fichier source finissant par **.cpp** (qui contient l'implémentation du code, i.e. le code à l'intérieur des fonctions qui ont été définies dans le fichier d'en tête).

**Le fichier d'en-tête (.h):** dans notre cas **Morse.h**

Contient l'ensemble des déclarations des méthodes de la classe **Morse** qui représentera la Bibliothèque.

*Avant la déclaration de la classe, on doit s'assurer que le fichier d'en tête ne sera pas victime d'une inclusion infinie.* Donc comme d'habitude en C, on ajoute une protection avec des directives de préprocesseur (e.g. **#ifndef...**).

Pour utiliser les constantes ou des fonctions faisant parti du core d'Arduino, il faut ajouter une **include** avant la déclaration de la classe =>

Code : C

```
#ifndef Morse_h
// si Morse_h n'est pas défini

#define Morse_h
// On le définit

// Code de la classe ICI

#endif // Fin si
```

Code : C

```
#include "WProgram.h"
```

Code : C

```
#ifndef Morse_h
#define Morse_h

#include "WProgram.h"

class Morse
{
public:
    Morse(int pin);
    void dot();
    void dash();
private:
    int _pin;
};

#endif
```

On ajoute aussi une propriété privée (**private**): **\_pin** qui contiendra le numéro du pin de la LED à utiliser.

De cette manière, lors de l'initialisation de l'objet **Morse**, on pourra à la fois mettre pin en mode **OUTPUT** et le stocker pour ne pas avoir à le redonner lorsqu'on fera un appel à une autre méthode.

Donc maintenant on peut s'attaquer à la déclaration de la classe.

Ce qui donne au final =>

**Le fichier source (.cpp) : dans notre cas Morse.cpp.**

Le fichier source contient l'implémentation des méthodes décrites dans le fichier d'en tête.

On arrive donc à ce code =>

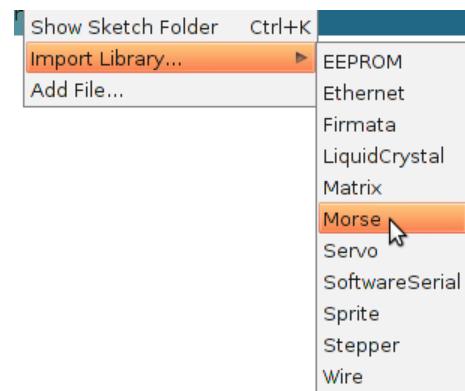
**Installer la bibliothèque:**

Dans le dossier du logiciel Arduino se trouve un sous-dossier nommé « libraries » où chaque sous-dossier représente une bibliothèque.

Pour ajouter la notre, on crée un dossier au nom de notre bibliothèque: e.g. dossier « Morse » et on y met les deux fichiers Morse.h et Morse.cpp.

**Utiliser la bibliothèque dans l'IDE:**

A partir de maintenant on peut voir la bibliothèque dans l'IDE au menu Sketch > Import Library si vous le relancez:



Code : C

```
/*
Morse.cpp - Bibliothèque pour code morse avec une DEL.

Récupéré du site officiel d'Arduino.
http://arduino.cc/en/Hacking/LibraryTutorial
Créé David A. Mellis, November 2, 2007.
Le code est du domaine public.
*/

#include "WProgram.h"
#include "Morse.h"

Morse::Morse(int pin)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
}

void Morse::dot()
{
  digitalWrite(_pin, HIGH);
  delay(250);
  digitalWrite(_pin, LOW);
  delay(250);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(1000);
  digitalWrite(_pin, LOW);
  delay(250);
}
```

Cliquer sur l'élément « MORSE » du menu, ce qui ajoutera le code nécessaire à l'utilisation de votre bibliothèque dans Arduino: #include <Morse.h>

### RÉSULTAT FINAL:

Grâce à la librairie, on peut maintenant simplifier notre code de départ. Ce qui nous permet d'avoir ce code =>

Ce code, évidemment, a le même effet que le code de départ.

Code : C

```
#include <Morse.h>

Morse morse(13);

void setup()
{
}

void loop()
{
    morse.dot(); morse.dot(); morse.dot();
    morse.dash(); morse.dash(); morse.dash();
    morse.dot(); morse.dot(); morse.dot();
    delay(3000);
}
```

La création de la bibliothèque est maintenant terminée.

Abusez de cette fonctionnalité pour simplifier votre code.

La création de bibliothèques vous assure la logique de votre code et diminue le temps de développement futur.

De plus, vous pouvez partager les bibliothèques pour en faire profiter la communauté !

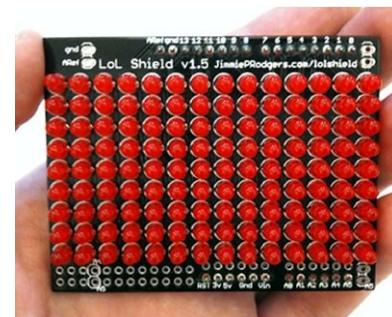
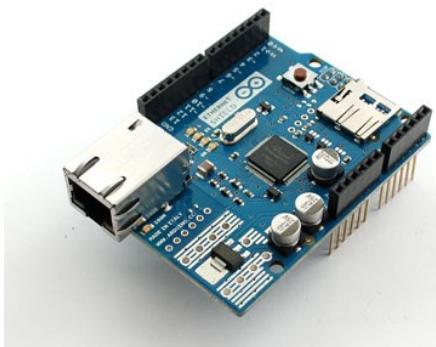
 RETOUR

## III-9 LES SHIELDS & LEUR CREATION

Un des intérêts majeurs de l'Arduino est qu'il existe aujourd'hui sur le marché, une multitude de cartes d'interfaces (~300), appelées Shields, capables de couvrir la majorité des besoins d'une application. Ces cartes de dimensions voisines de l'Arduino s'enfichent directement sur les connecteurs périphériques de cette dernière. Ils suivent la même philosophie qu'Arduino: faciles à monter, et peu coûteux à produire.

On trouve de nombreuses cartes entre 15 et 30€ pièce qui ajoutent des fonctionnalités intéressantes:

- Ethernet,
- GPS,
- transmissions sans fil (Xbee),
- interfaces LCD graphique couleur, monochrome, ou texte,
- clavier,
- Game shield,
- matrices de leds...



<http://shieldlist.org/>

## FABRIQUER CES SHIELDS:

<http://aaroneiche.com/2010/06/24/a-beginners-guide-to-making-an-arduino-shield-pcb/>

<http://www.krisbarrett.com/2008/09/03/make-a-custom-arduino-shield/>

## STANDARDISATION DES SHIELDS:

<http://www.practicalarduino.com/news/id/661>

<http://www.practicalmaker.com/blog/arduino-shield-design-standards>

## LOGICIELS OPEN-SOURCE:

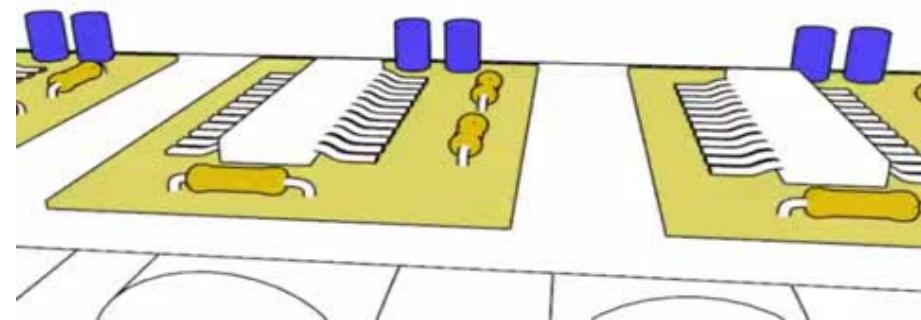
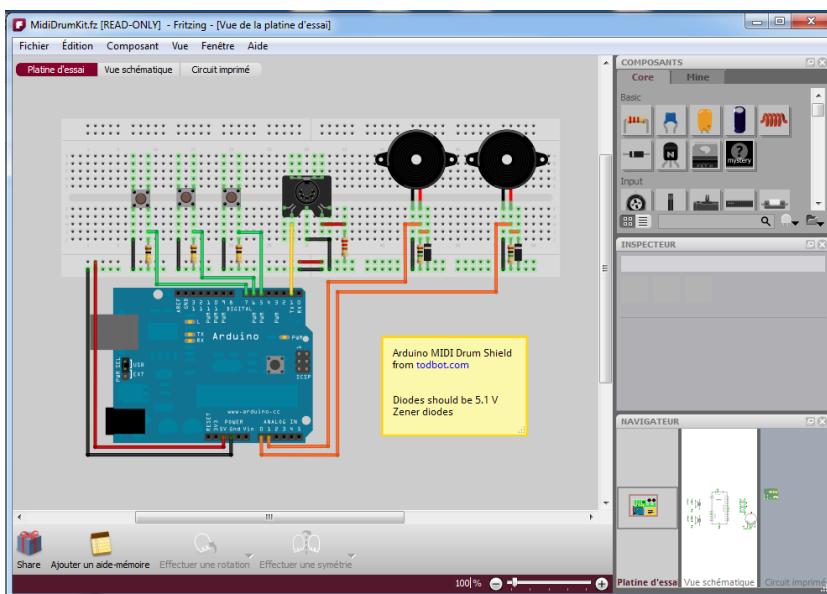
ExpressPCB: <http://www.expresspcb.com/>

Fritzing: <http://fritzing.org/>

KiCAD: [http://www.lis.inpg.fr/realise\\_au\\_lis/kicad/](http://www.lis.inpg.fr/realise_au_lis/kicad/)

TCI:<http://b.urbani.free.fr/pagetci/tci.htm>

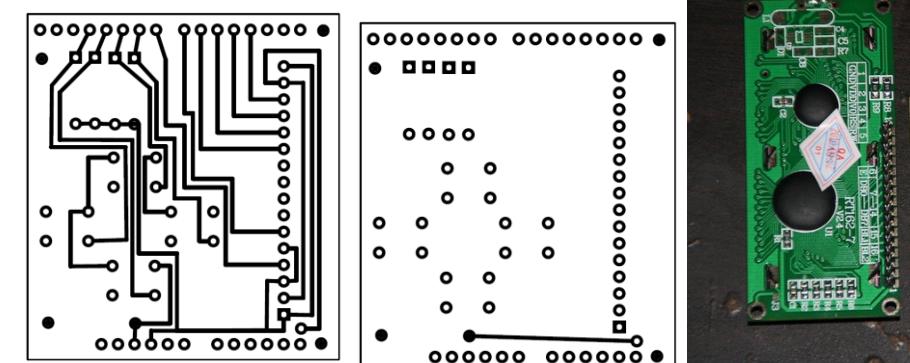
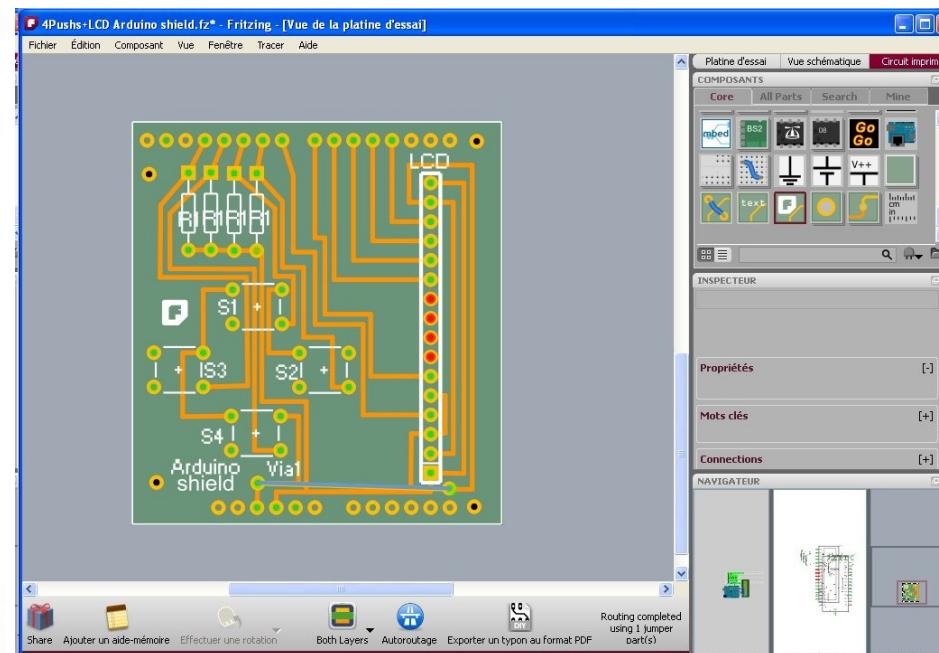
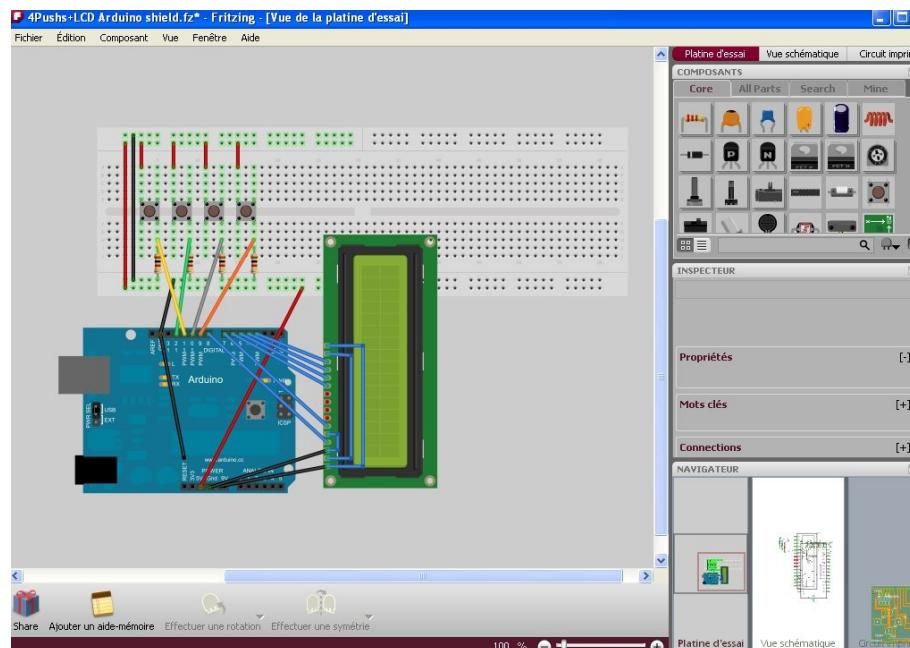
Fritzing permet de dessiner son circuit de façon graphique, schématique ou électronique



# EXAMPLE

Exemple: LCD avec 4 boutons

<http://fritzing.org/projects/arduino-lcd-4-push-buttons/>



**expresspcb™**

Introduction to ExpressPCB  
How it Works  
How Much it Costs  
Free CAD Software  
PC Board Manufacturing  
Download ExpressPCB  
Customer Feedback  
Tips for Designing PCBs  
Resources for Engineers  
Our Mission  
How to Contact Us

**\$51<sup>For 3</sup> PCBS**  
FREE Layout Software!  
FREE Schematic Software!

**ExpressPCB makes prototyping electronics EASY, FAST & AFFORDABLE!**

Copyright © 2010, Express PCB

Free CAD Software & Low cost PCBs

download design order receive

**Intro to ExpressPCB**

> Our Free PCB software is a snap to learn and use. For the first time, designing circuit boards is simple for the beginner and efficient for the professional.

> Our board manufacturing service makes top quality two and four layer PCBs. Use our MiniBoard service and pay only \$51 for three boards (plus shipping).

Learning the ExpressPCB software is fast because of its familiar user interface.

Best of all, its Free!

Our PCBs are both excellent quality and very affordable.

Most two layer boards are shipped the next business day!

EAGLE PCB DESIGN

f EN / DE

**Powerful PCB Design**

PCB design tailored to meet the needs of professional engineers, makers and students!

Download EAGLE 7.6  
Win / Linux / OS X

Home Tour Pricing Support Resources Blog

<http://www.expresspcb.com/>

## Schematic Editor

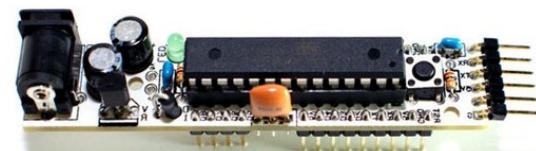
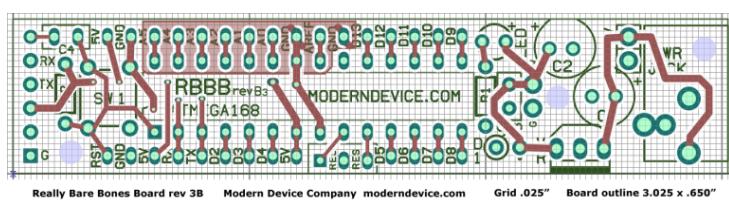
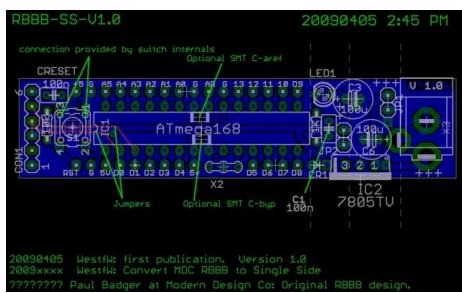
EAGLE's easy-to-use schematic editor allows you to create an easy-to-read representation of your electronics design with zero complexity. Be productive in minutes with the easiest, most intuitive schematic tool available!

Great schematic capture is essential for documenting your design.

This website uses cookies to improve your experience.

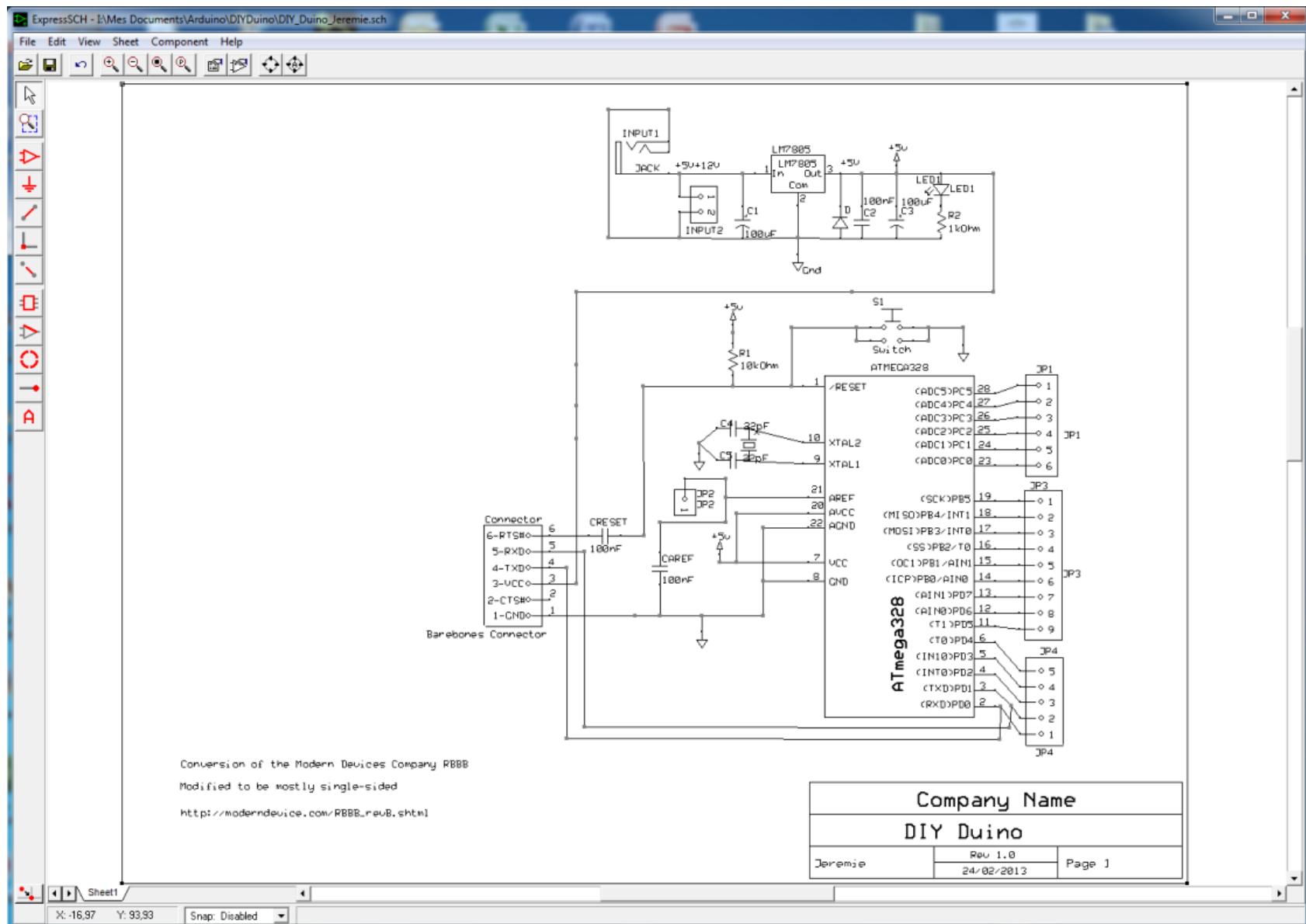
OK

<https://cadsoft.io/>

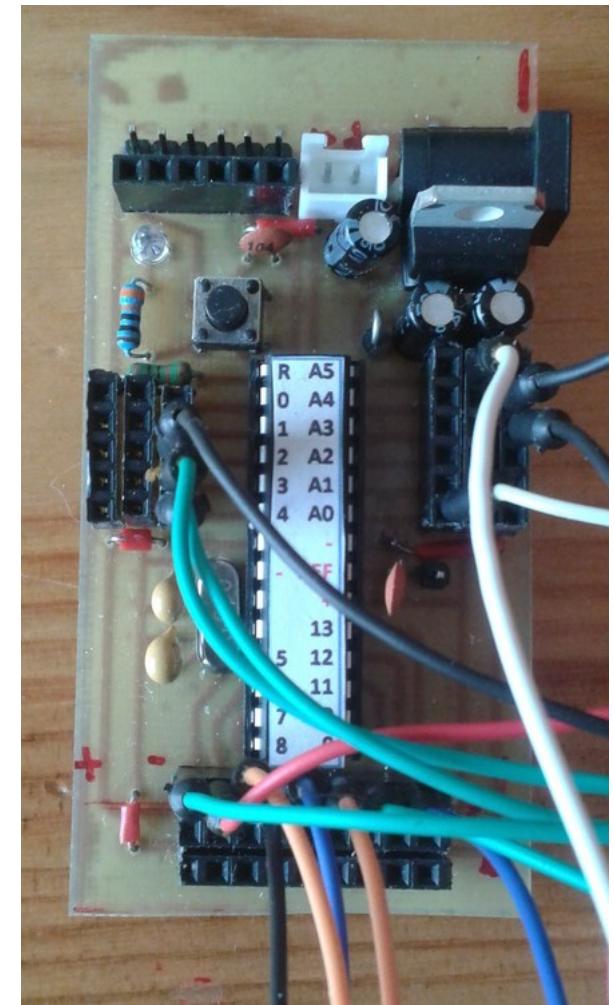
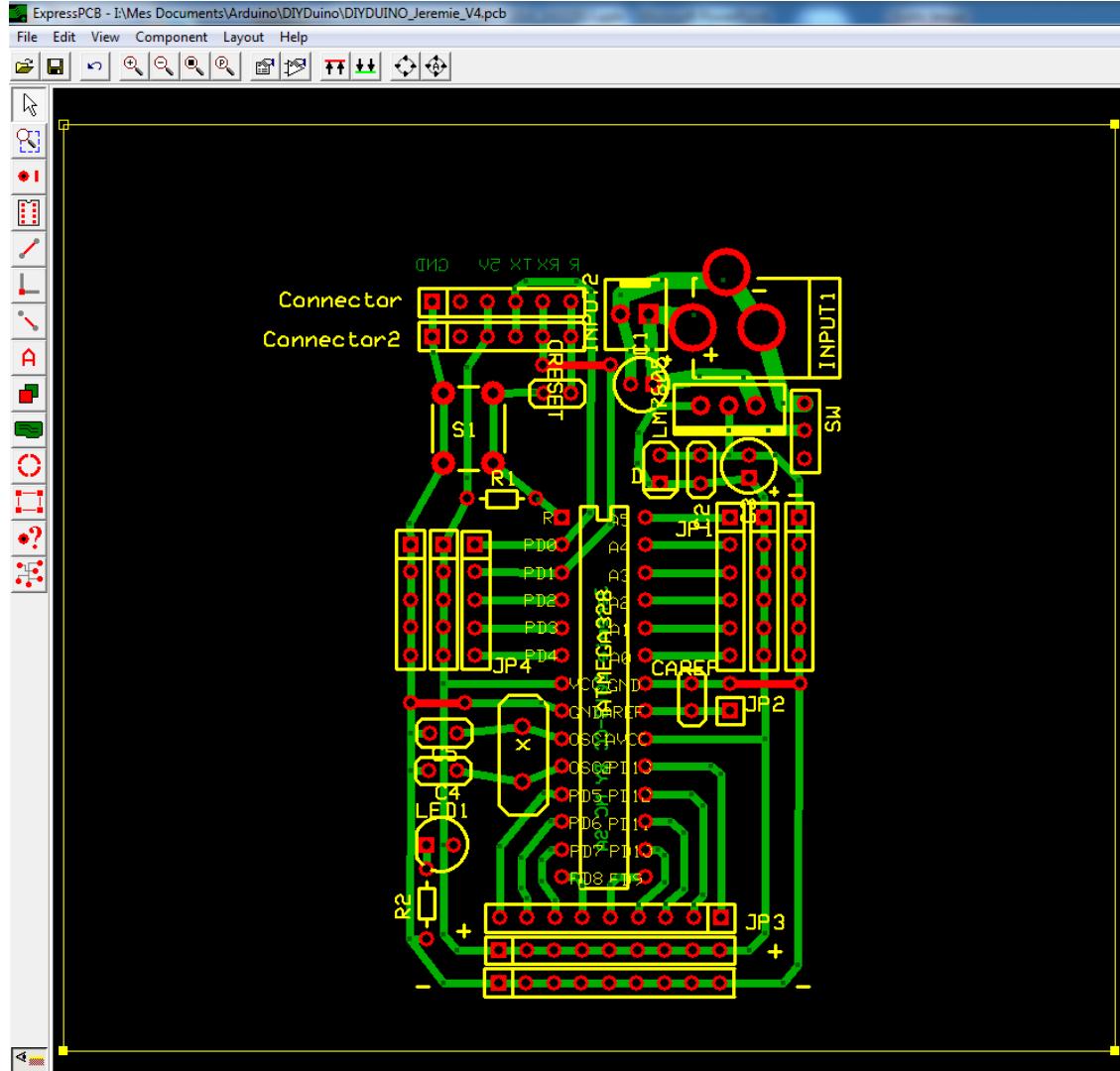


RETOUR

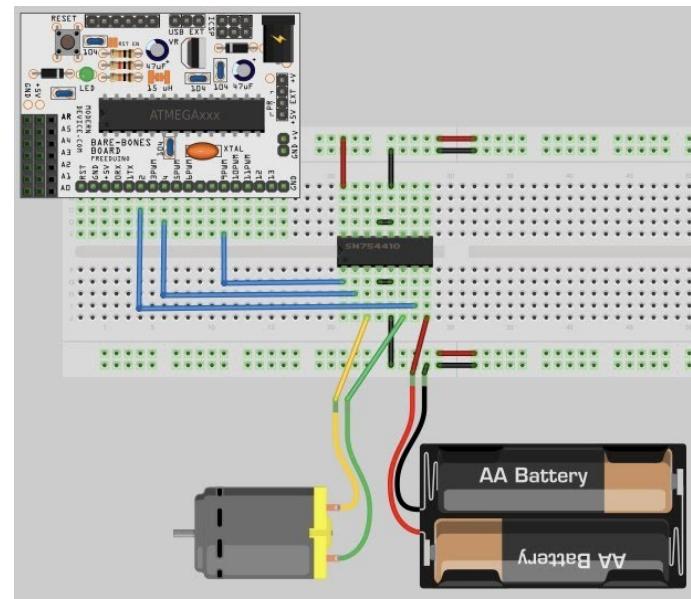
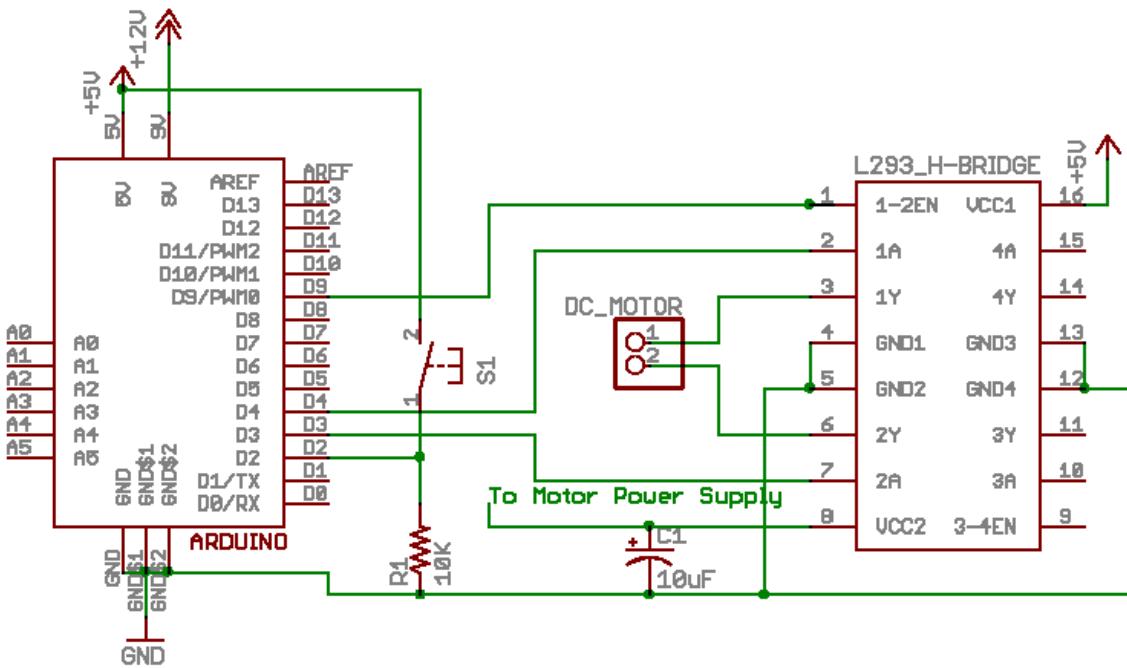
## EXPRESS SCH: ARDUINO LIKE



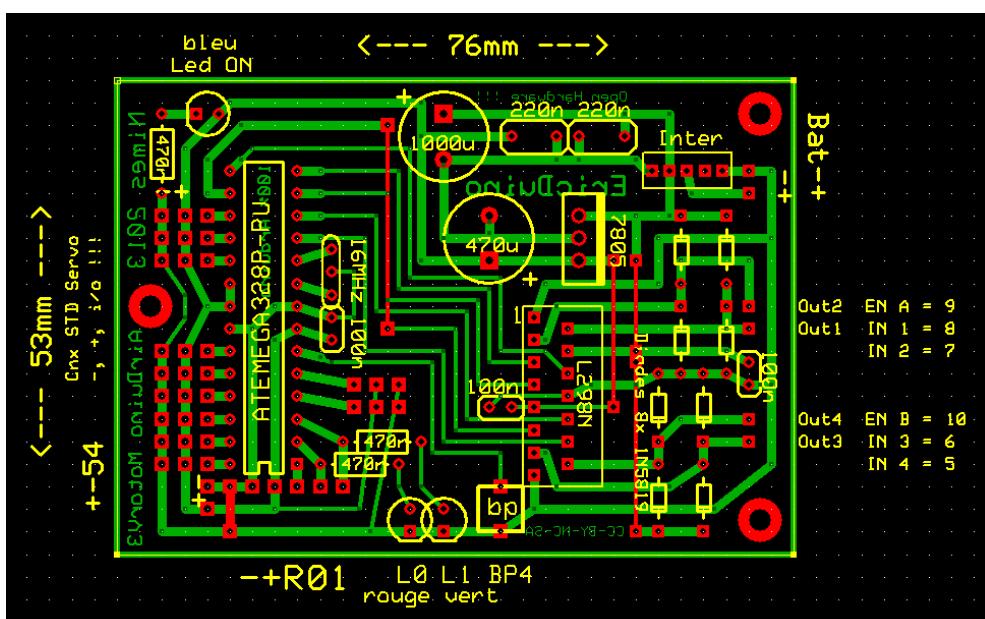
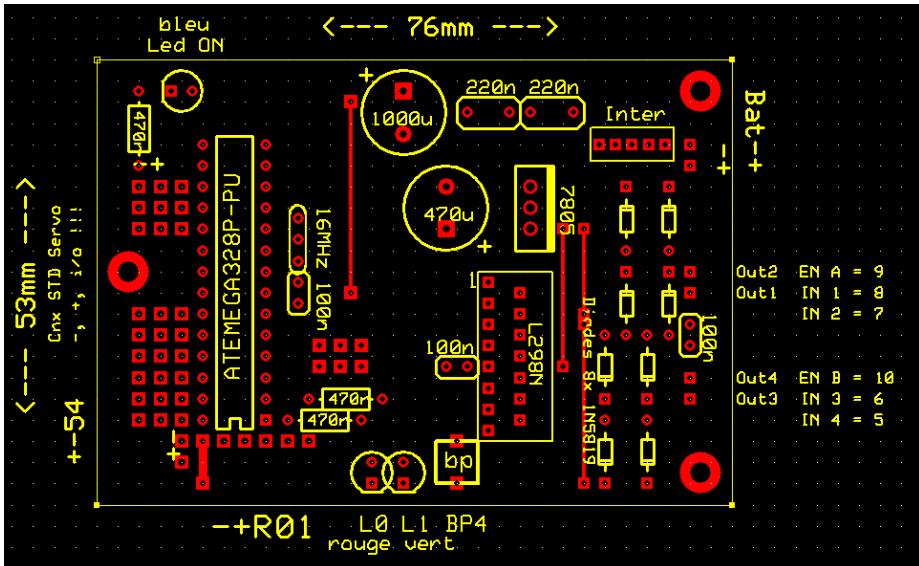
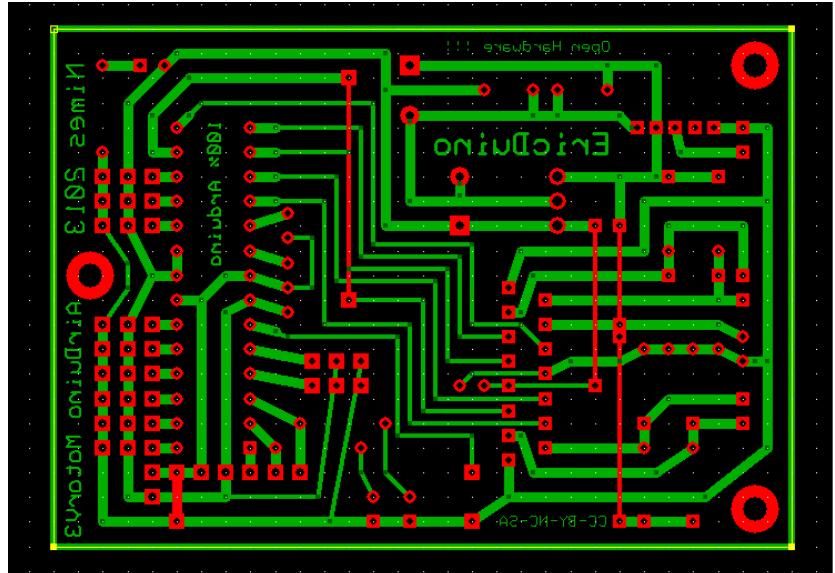
## EXPRESS PCB: ARDUINO LIKE



<http://www.expresspcb.com/>

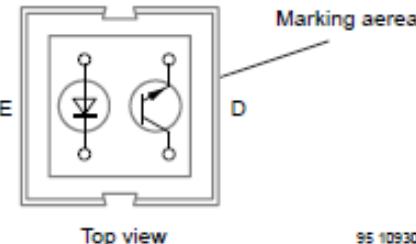


# CARTE ARDUINO + PONT H

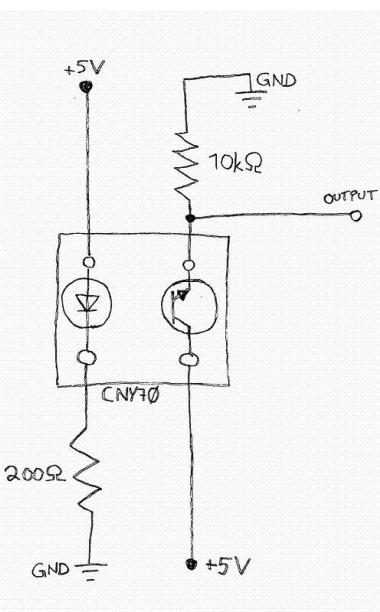


RETOUR

## OPTOCOUPLEUR: CNY70



95 10930



### Electrical Characteristics ( $T_{amb} = 25^\circ\text{C}$ )

#### Input (Emitter)

Parameter	Test Conditions	Symbol	Min.	Typ.	Max.	Unit
Forward voltage	$I_F = 50 \text{ mA}$	$V_F$		1.25	1.6	V

#### Output (Detector)

Parameter	Test Conditions	Symbol	Min.	Typ.	Max.	Unit
Collector emitter voltage	$I_C = 1 \text{ mA}$	$V_{CEO}$	32			V
Emitter collector voltage	$I_E = 100 \mu\text{A}$	$V_{ECO}$	5			V
Collector dark current	$V_{CE} = 20 \text{ V}, I_F = 0, E = 0$	$I_{CEO}$			200	nA

#### Coupler

Parameter	Test Conditions	Symbol	Min.	Typ.	Max.	Unit
Collector current	$V_{CE} = 5 \text{ V}, I_F = 20 \text{ mA}, d = 0.3 \text{ mm}$ (figure 1)	$I_C^{(1)}$	0.3	1.0		mA
Cross talk current	$V_{CE} = 5 \text{ V}, I_F = 20 \text{ mA}$ (figure 1)	$I_{Cx}^{(2)}$			600	nA
Collector emitter saturation voltage	$I_F = 20 \text{ mA}, I_C = 0.1 \text{ mA}, d = 0.3 \text{ mm}$ (figure 1)	$V_{CESat}^{(1)}$			0.3	V
① Measured with the 'Kodak neutral test card', white side with 90% diffuse reflectance						
② Measured without reflecting medium						

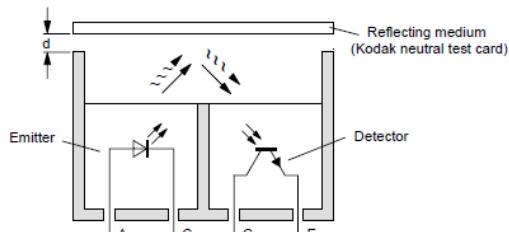


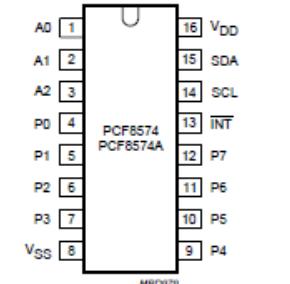
Figure 1. Test circuit

## EXPANDER I2C: PCF8574

Remote 8-bit I/O expander for I2C-bus

### 5 PINNING

SYMBOL	PIN		DESIGNATION
	DIP16; SO16	SSOP20	
A0	1	6	address input 0
A1	2	7	address input 1
A2	3	9	address input 2
P0	4	10	quasi-bidirectional I/O 0
P1	5	11	quasi-bidirectional I/O 1
P2	6	12	quasi-bidirectional I/O 2
P3	7	14	quasi-bidirectional I/O 3
V <sub>SS</sub>	8	15	supply ground
P4	9	16	quasi-bidirectional I/O 4
P5	10	17	quasi-bidirectional I/O 5
P6	11	19	quasi-bidirectional I/O 6
P7	12	20	quasi-bidirectional I/O 7
INT	13	1	interrupt output (active LOW)
SCL	14	2	serial clock line
SDA	15	4	serial data line
V <sub>DD</sub>	16	5	supply voltage
n.c.	—	3	not connected
n.c.	—	8	not connected
n.c.	—	13	not connected
n.c.	—	18	not connected



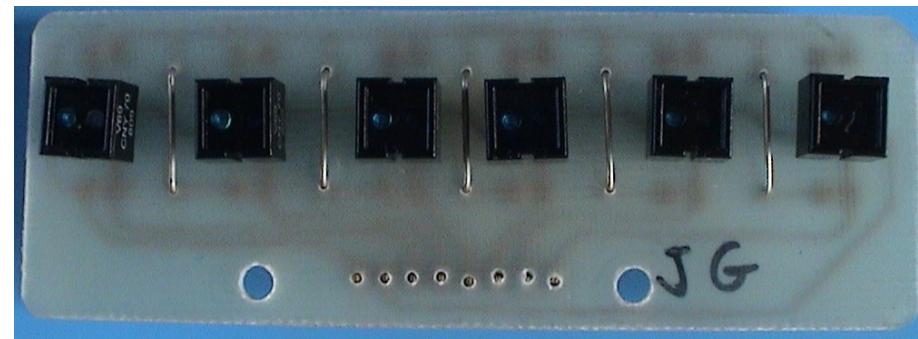
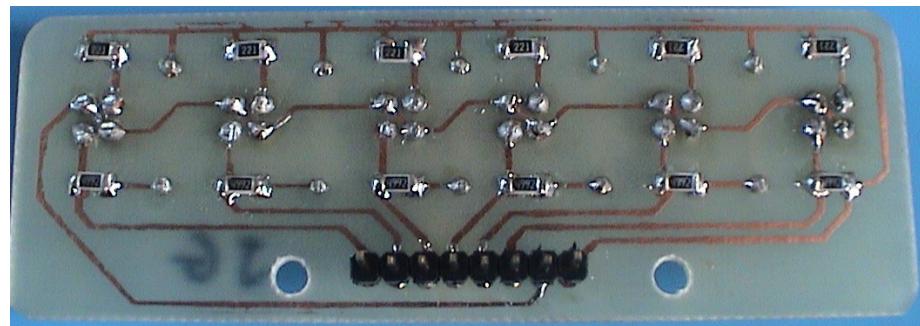
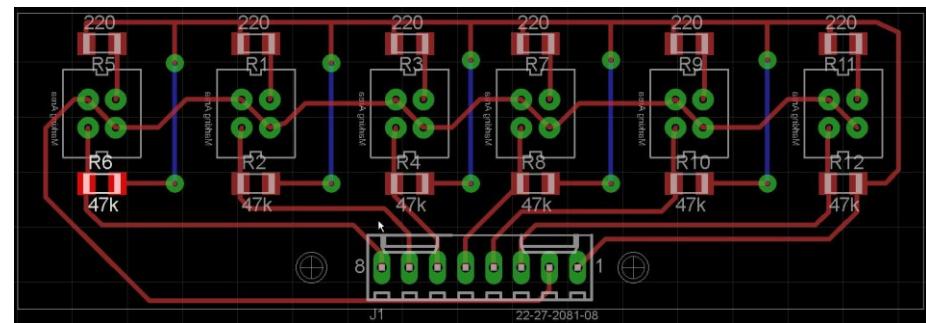
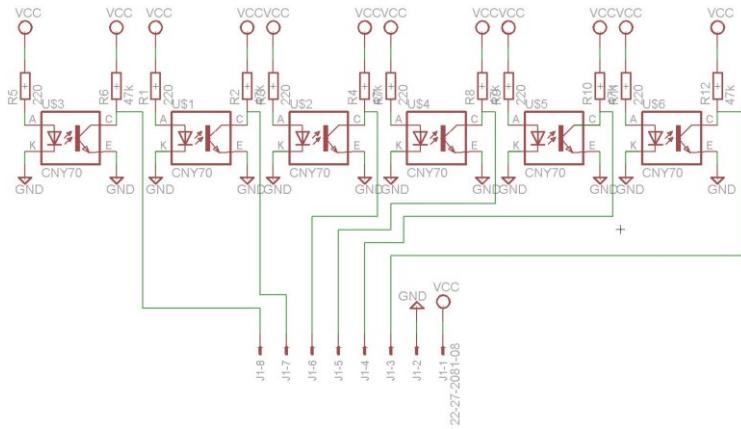
a PCF8574

Each of the PCF8574's eight I/Os can be independently used as an input or output. Input data is transferred from the port to the microcontroller by the READ mode (see Fig.12). Output data is transmitted to the port by the WRITE mode (see Fig.11).



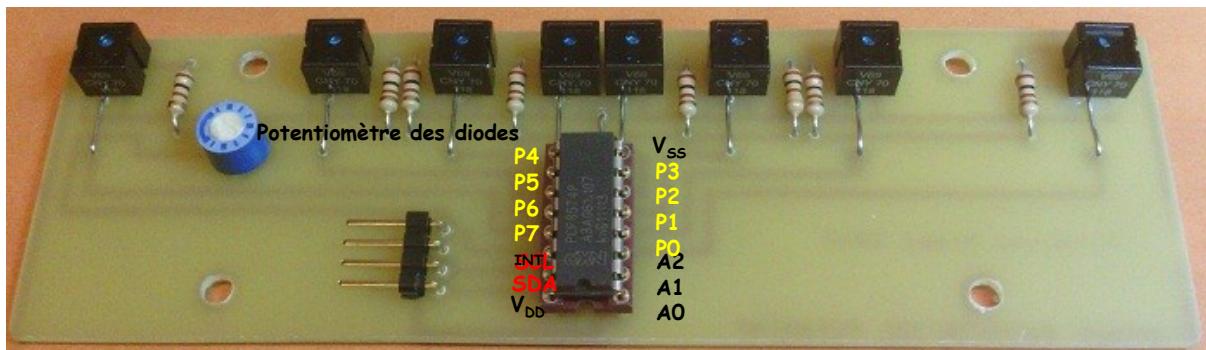
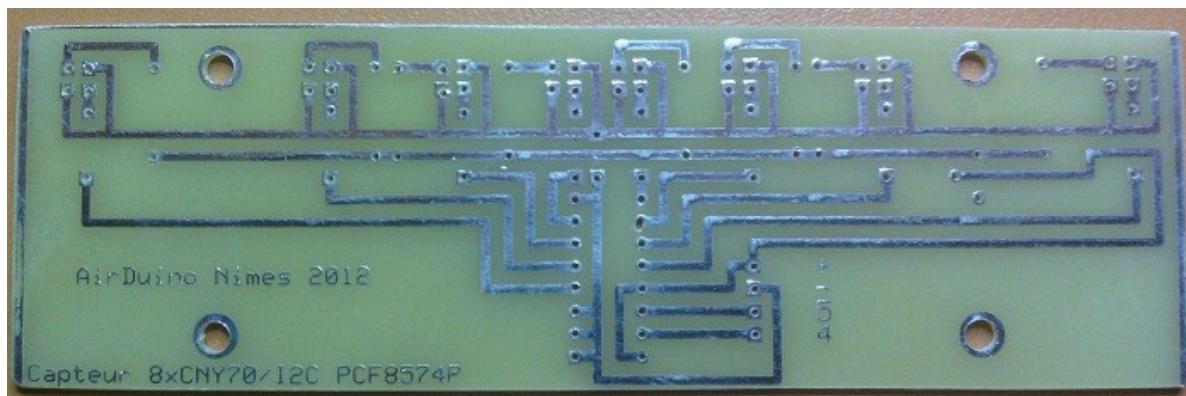
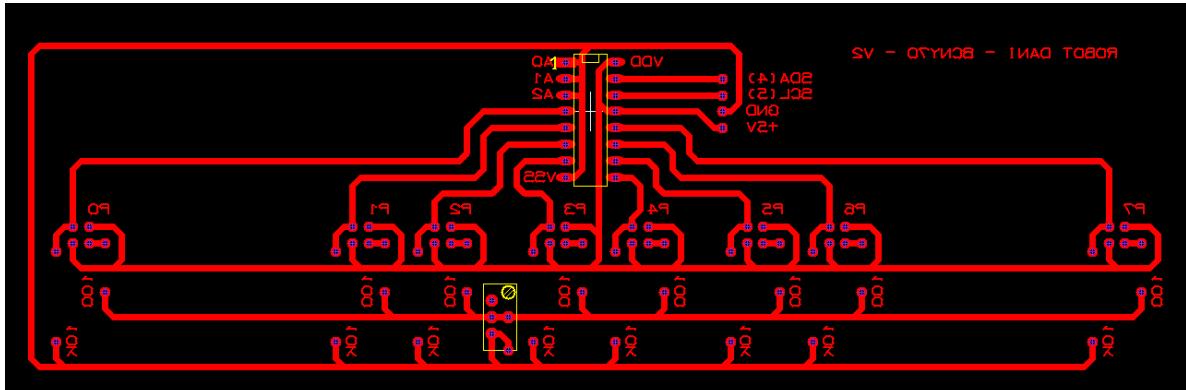
# EXEMPLE DE CARTES: SUIVEUR DE LIGNES - 1

## SANS EXPANDER I2C: PCF8574



# EXEMPLE DE CARTES: SUIVEUR DE LIGNES - 2

## AVEC EXPANDER I2C: PCF8574

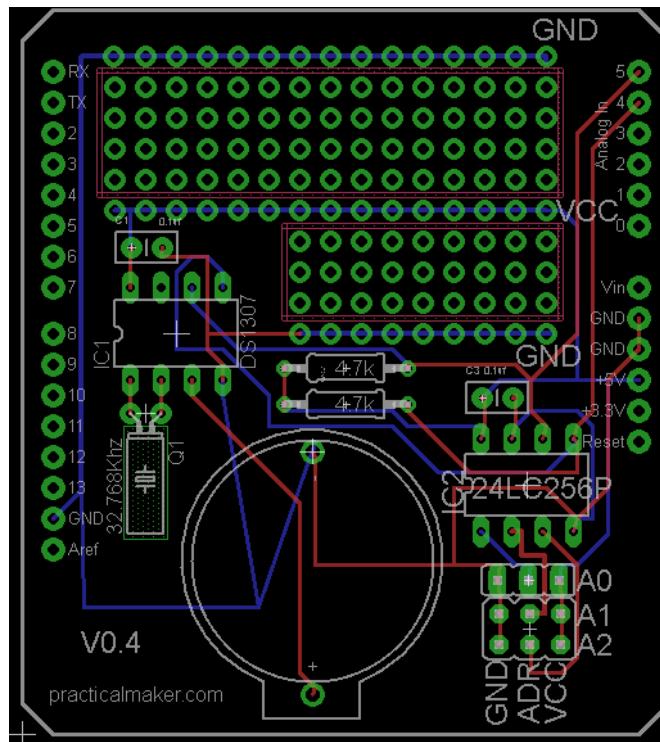


Concevoir un shield nécessite une standardisation qui rendra l'ensemble du système plus modulaire et ouvrira de nouveaux champs de possibilités pour l'Arduino.

L'idée est simple: lors de la conception d'un shield, une certaine logique doit être respectée

Le tableau suivant sont les directives pour l'affectation des broches lors de la conception d'un shield.

Pin	Reserved For
Analog 0	User Selectable Input
Analog 1	User Selectable Input
Analog 2	User Selectable Input
Analog 3	User Selectable Input
Analog 4	I2C
Analog 5	I2C
Digital 0	RX
Digital 1	TX
Digital 2	OneWire
Digital 3	PWM / Digital I/O / I2C Interrupt
Digital 4	CS Select
Digital 5	PWM / Digital I/O
Digital 6	PWM / Digital I/O
Digital 7	CS Select
Digital 8	CS Select
Digital 9	PWM / Digital I/O
Digital 10	PWM / Digital I/O
Digital 11	SPI
Digital 12	SPI
Digital 13	SPI

Analog 0	User Selectable Input
Analog 1	User Selectable Input
Analog 2	User Selectable Input
Analog 3	User Selectable Input
Analog 4	I2C
Analog 5	I2C
Digital 0	RX
Digital 1	TX
Digital 2	OneWire
Digital 3	PWM / Digital I/O / I2C Interrupt
Digital 4	CS Select
Digital 5	PWM / Digital I/O
Digital 6	PWM / Digital I/O
Digital 7	CS Select
Digital 8	CS Select
Digital 9	PWM / Digital I/O
Digital 10	PWM / Digital I/O
Digital 11	SPI
Digital 12	SPI
Digital 13	SPI


<http://www.practicalmaker.com/blog/arduino-shield-design-standards>

# IV-1 POUR ALLER PLUS LOIN: PROCESSING

# PROCESSING



« PROCESSING » rend la programmation JAVA aussi amusante et facile que la programmation des AVR avec l'Arduino

- Il est très utilisé pour connecter des Arduino aux PC
- Il est totalement OPEN-SOURCE comme l'Arduino

Les sketches Processing sont très similaires aux sketches Arduino

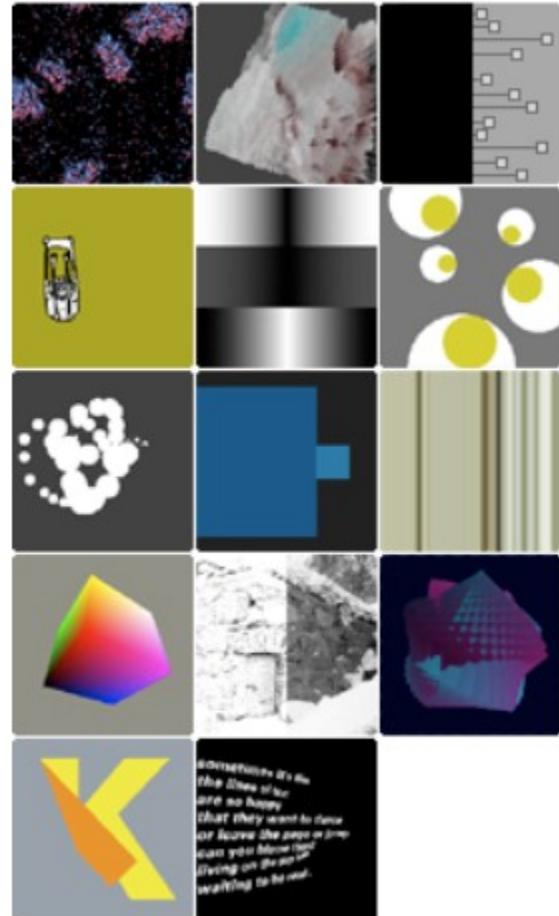
- **setup()** – mise en place du sketch
- **draw()** – comme **loop()** dans Arduino (appelée régulièrement)
- Les autres fonctions dépendent des librairies

```
Bounce | Processing 1.5.1
File Edit Sketch Tools Help
STANDARD
Bounce
background(102);

// Update the position of the shape
xpos = xpos + ( xspeed * xdirection );
ypos = ypos + ( yspeed * ydirection );

// Test to see if the shape exceeds the boundaries of the screen
// If it does, reverse its direction by multiplying by -1
if (xpos > width-size || xpos < 0) {
  xdirection *= -1;
}
if (ypos > height-size || ypos < 0) {
  ydirection *= -1;
}

// Draw the shape
ellipse(xpostsize/2, ypos+size/2, size, size);
}
```



<http://processing.org/learning/gettingstarted/>

[http://wiki.processing.org/w/Main\\_Page](http://wiki.processing.org/w/Main_Page)

[http://www.ecole-art-aix.fr/rubrique.php?id\\_rubrique=81](http://www.ecole-art-aix.fr/rubrique.php?id_rubrique=81)

Téléchargez et installez PROCESSING:

<http://processing.org/download/>

```
/** Bounce.
 * When the shape hits the edge of the window, it reverses its direction.
 */
int size = 60;      // Width of the shape
float xpos, ypos;  // Starting position of shape
float xspeed = 2.8; // Speed of the shape
float yspeed = 2.2; // Speed of the shape
int xdirection = 1; // Left or Right
int ydirection = 1; // Top to Bottom
void setup()
{
size(640, 200);
noStroke();
frameRate(30);
smooth();
// Set the starting position of the shape
xpos = width/2;
ypos = height/2;
}

void draw()
{
background(102);

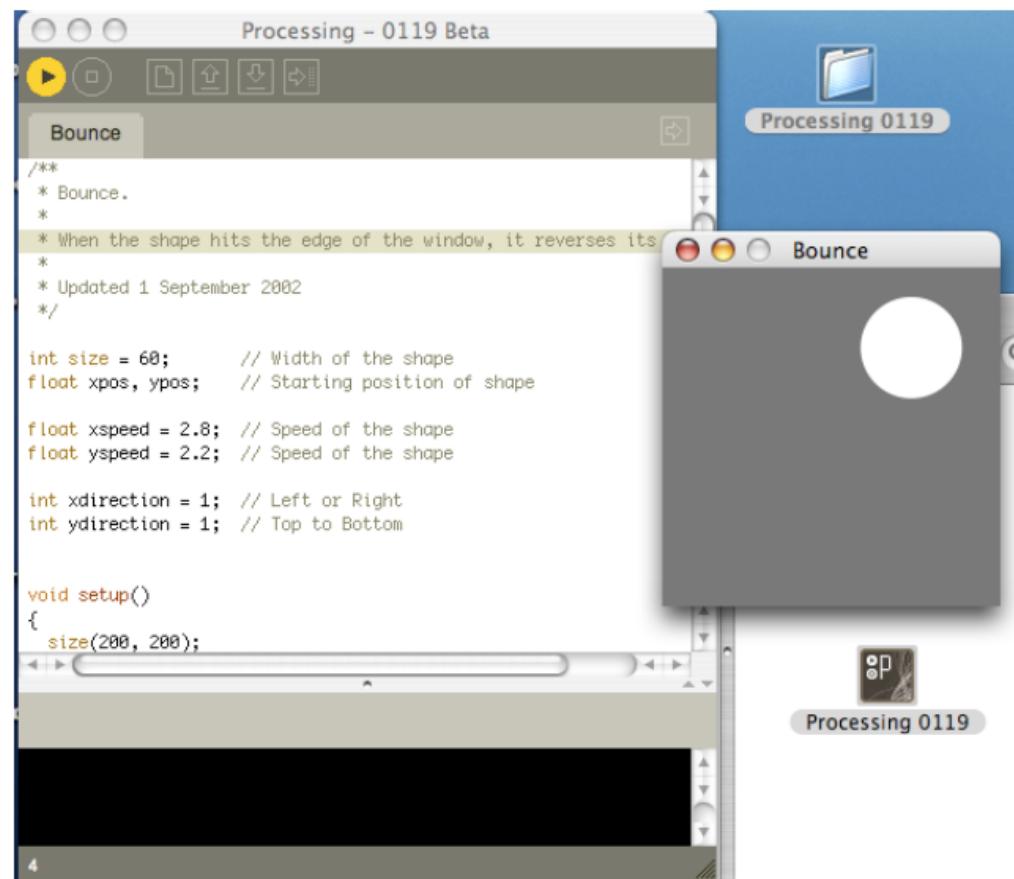
// Update the position of the shape
xpos = xpos + ( xspeed * xdirection );
ypos = ypos + ( yspeed * ydirection );

// Test to see if the shape exceeds the boundaries of the screen
// If it does, reverse its direction by multiplying by -1
if (xpos > width-size || xpos < 0) {
xdirection *= -1;
}
if (ypos > height-size || ypos < 0) {
ydirection *= -1;
}
// Draw the shape
ellipse(xpos+size/2, ypos+size/2, size, size);
}
```

1 - Chargez l'exemple: TOPICS/MOTION/BOUNCE

2 – Pressez RUN

=> vous venez de créer une applet JAVA



- Processing et Arduino parlent tous les deux le “SERIE”
- Mais, un seul programme par port Série possible: ainsi, fermez le moniteur Serie Arduino quand vous vous connectez via Processing, et vice-versa.
- Processing possède une librairie “Serie” pour parler à Arduino:  
`port = new Serial(this,"mon_nom_port",19200)`  
`port.read(), port.write(), port.available(), etc.`  
`serialEvent() { }`

```
import processing.serial.*;

String portname = "/dev/tty.usbserial-A4001qa8"; // or "COM8"
Serial port; // Create object from Serial class

int val=100; // Data received from the serial port, with an initial value

void setup()
{
    // Open the port the board is connected to
    port = new Serial(this, portname, 19200);
}

void draw()
{
    if (port.available() > 0) { // If data is available,
        val = port.read(); // read it and store it in val
    }
}
```

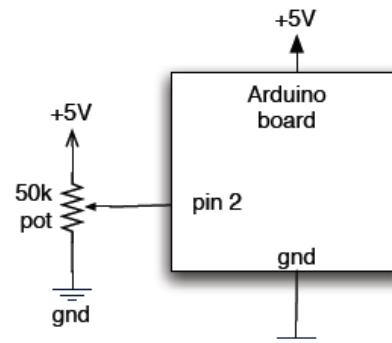
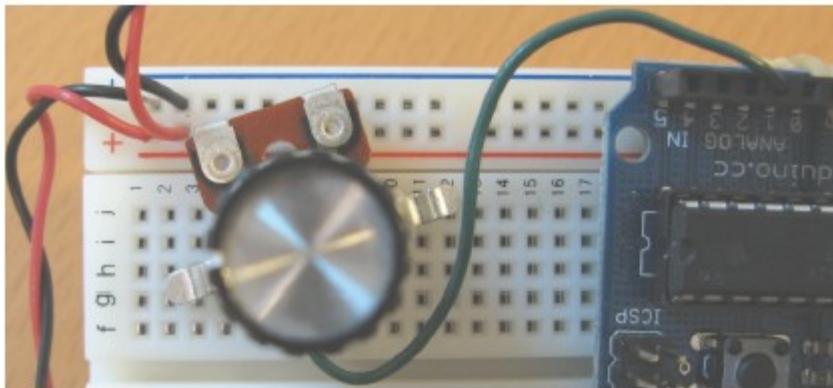
Assurez-vous de mettre le bon nom de port Série !

#### 4 étapes:

1. Chargez la librairie
2. Définir le nom du port: portname
3. Ouvrir le port: port=new Serial(...);
4. Lire/Ecrire sur port

# ARDUINO PARLE A PROCESSING

Créez un programme Arduino qui lit la valeur d'un potentiomètre et l'envoie à Processing sous la forme d'un nombre entre 0 et 255:



## Arduino Sketch:

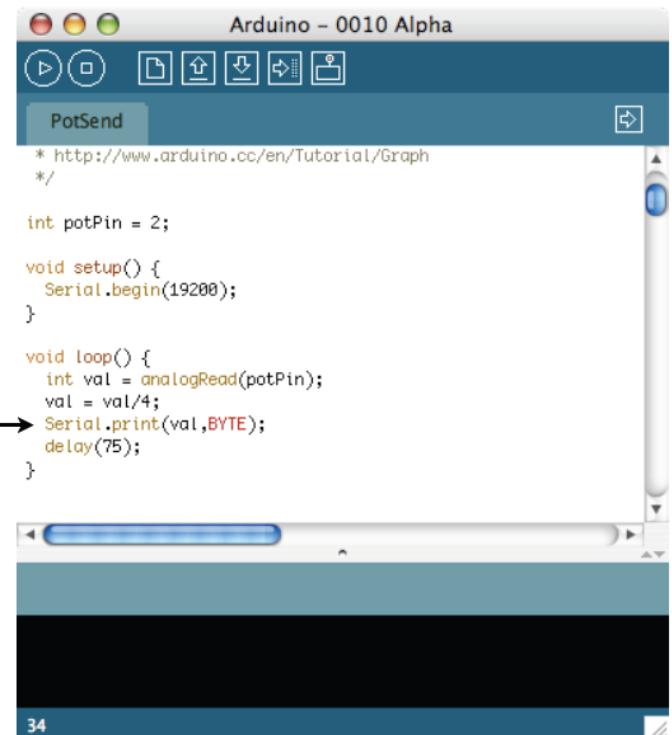
```

/* PotSend
 * Send analog values from pots or similar over the serial port
 * Also see: http://www.arduino.cc/en/Tutorial/Graph
 */
int potPin = 2;

void setup() {
  Serial.begin(19200);
}

void loop() {
  int val = analogRead(potPin);
  val = val/4;
  Serial.print(val,BYTE);
  delay(100);
}
  
```

Note: n'envoyez pas la valeur comme texte ASCII, mais plutôt comme un nombre binaire (les BYTE sont plus faciles à manipuler dans Processing)



```
/** Collision (Pong).
 * Move the mouse up and down to move the
paddle.
 * Modified to use Serial port by Tod E. Kurt, 2007
 * Updated 13 January 2003 by K Pfeiffer
 */

import processing.serial.*;
```

```
String portname = "/dev/tty.usbserial-A4001qa8";
```

```
// or "COM8"
```

```
Serial port; // Create object from Serial class
```

```
// Global variables for the ball
```

```
float ball_x;
float ball_y;
float ball_dir = 1;
float ball_size = 5; // Radius
float dy = 0; // Direction
```

```
// Global variables for the paddle
```

```
int paddle_width = 5;
int paddle_height = 20;
int paddle_pos; // new position
int paddle_ppos; // last position
int dist_wall = 15;
```

```
void setup()
```

```
{
size(255, 255);
rectMode(CENTER_RADIUS);
ellipseMode(CENTER_RADIUS);
noStroke();
smooth();
ball_y = height/2;
ball_x = 1;
```

**// Open the port that the board is connected to and use the same speed (19200 bps)**

```
port = new Serial(this, portname, 19200);
```

```
}
```

```
void draw()
{
background(51);
ball_x += ball_dir * 1.0;
ball_y += dy;
if(ball_x > width+ball_size) {
    ball_x = -width/2 - ball_size;
    ball_y = random(0, height);
    dy = 0;
}
if(port.available() > 0) { // If data is available,
    paddle_ppos = paddle_pos; // save old position
    paddle_pos = port.read(); // read it and store it as the new pos
    paddle_pos = paddle_pos *2; //pour avoir les valeurs de 0 à 255
    println(paddle_pos);
}
// Constrain paddle to screen
float paddle_y = constrain(paddle_pos, paddle_height, height-paddle_height);

// Test to see if the ball is touching the paddle
float py = width-dist_wall-paddle_width-ball_size;
if(ball_x == py
    && ball_y > paddle_y - paddle_height - ball_size
    && ball_y < paddle_y + paddle_height + ball_size) {
    ball_dir *= -1;
    if(paddle_pos != paddle_ppos) {
        dy = (paddle_pos - paddle_ppos)/2.0;
        if(dy > 5) { dy = 5; }
        if(dy < -5) { dy = -5; }
    }
}
// If ball hits paddle or back wall, reverse direction
if(ball_x < ball_size && ball_dir == -1) {
    ball_dir *= -1;
}
// If the ball is touching top or bottom edge, reverse direction
if(ball_y > height-ball_size) {
    dy = dy * -1;
}
if(ball_y < ball_size) {
    dy = dy * -1;
}
```

<http://moodle.insa-toulouse.fr/course/view.php?id=494>

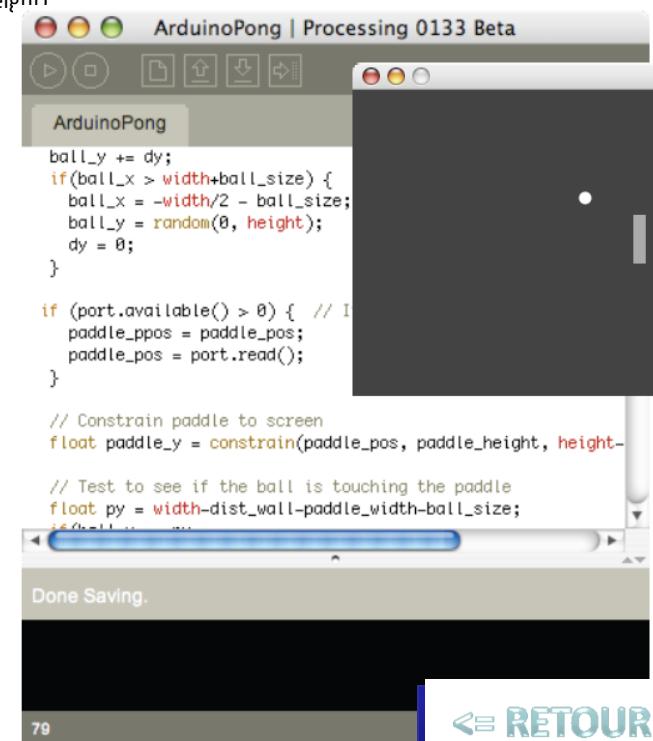
```
// Draw ball
fill(255);
ellipse(ball_x, ball_y, ball_size, ball_size);

// Draw the paddle
fill(153);
rect(width-dist_wall, paddle_y, paddle_width, paddle_height);
}
```

## La base du jeu de pong

Le potentiomètre contrôle la position de la raquette

Ajoutez un potentiomètre et quelques lignes de codes et vous aurez un jeu à deux joueurs



RETOUR

## Sketch ARDUINO:

```
#include <math.h>

#include "Wire.h"
#include "WiiChuck.h"
//#include "nunchuck_funcs.h"

#define MAXANGLE 90
#define MINANGLE -90

WiiChuck chuck = WiiChuck();
int angleStart, currentAngle;
int tillerStart = 0;
double angle;

void setup() {
  //nunchuck_init();
  Serial.begin(115200);
  chuck.begin();
  chuck.update();
  //chuck.calibrateJoy();
}

void loop() {
  delay(20);
  chuck.update();

  Serial.print(chuck.readRoll());
  Serial.print(", ");
  Serial.print(chuck.readPitch());
  Serial.print(", ");

  Serial.print((int)chuck.readAccelX());
  Serial.print(", ");
  Serial.print((int)chuck.readAccelY());
  Serial.print(", ");

  Serial.print((int)chuck.readAccelZ());
  Serial.println();
}
```

<http://arduino.cc/playground/Main/WiiChuckClass>

## Call WiiChuck.h:

/\* Nunchuck -- Use a Wii Nunchuck \* Tim Hirzel <http://www.growdown.com> \*  
notes on Wii Nunchuck Behavior.

This library provides an improved derivation of rotation angles from the nunchuck accelerometer data. The biggest difference over existing libraries (that I know of) is the full 360 degrees of Roll data  
from the combination of the x and z axis accelerometer data using the math library atan2.

It is accurate with 360 degrees of roll (rotation around axis coming out of the c button, the front of the wii), and about 180 degrees of pitch (rotation about the axis coming out of the side of the wii). (read more below)

In terms of mapping the wii position to angles, its important to note that while the Nunchuck sense Pitch, and Roll, it does not sense Yaw, or the compass direction. This creates an important disparity where the nunchuck only works within one hemisphere. At a result, when the pitch values are less than about 10, and greater than about 170, the Roll data gets very unstable. essentially, the roll data flips over 180 degrees very quickly. To understand this property better, rotate the wii around the axis of the joystick. You see the sensor data stays constant (with noise). Because of this, it cant know the difference between arriving upside via 180 degree Roll, or 180 degree pitch. It just assumes its always 180 roll. \*

\* This file is an adaptation of the code by these authors:

\* Tod E. Kurt, <http://todbot.com/blog/>

\* The Wii Nunchuck reading code is taken from Windmeadow Labs

\* <http://www.windmeadow.com/node/42>

\*/

```
#ifndef WiiChuck_h
#define WiiChuck_h
#include "WProgram.h"
#include <Wire.h>
#include <math.h>
```

```
// these may need to be adjusted for each nunchuck for calibration
#define ZEROX_510
#define ZEROY_490
#define ZEROZ_460
#define RADIUS 210 // probably pretty universal
```

```
#define DEFAULT_ZERO_JOY_X 124
#define DEFAULT_ZERO_JOY_Y 132
```

```
class WiiChuck {
private:
    byte cnt;
    uint8_t status[6]; // array to store wiichuck output
    byte averageCounter;
    //int accelArray[3][AVERAGE_N]; // X,Y,Z
    int i;
    int total;
    uint8_t zeroJoyX; // these are about where mine are
    uint8_t zeroJoyY; // use calibrateJoy when the stick is at zero to correct
    int lastJoyX;
    int lastJoyY;
    int angles[3];
}
```

```
boolean lastZ, lastC;
```

```
public:
    byte joyX;
    byte joyY;
    boolean buttonZ;
    boolean buttonC;
```

```
void begin()
{
    Wire.begin();
    cnt = 0;
    averageCounter = 0;
    // instead of the common 0x40 -> 0x00 initialization, we
    // use 0xF0 -> 0x55 followed by 0xFB -> 0x00.
    // this lets us use 3rd party nunchucks (like cheap $4 ebay ones)
    // while still letting us use official ones.
    // only side effect is that we no longer need to decode bytes in _nunchuk_decode_byte
    // see http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?num=1264805255
    //
    Wire.beginTransmission(0x52); // device address
    Wire.send(0xF0);
    Wire.send(0x55);
    Wire.endTransmission();
    delay(1);
    Wire.beginTransmission(0x52);
    Wire.send(0xFB);
    Wire.send(0x00);
    Wire.endTransmission();
    update();
    for (i = 0; i<3;i++) {
        angles[i] = 0;
    }
    zeroJoyX = DEFAULT_ZERO_JOY_X;
    zeroJoyY = DEFAULT_ZERO_JOY_Y;
}

void calibrateJoy() {
    zeroJoyX = joyX;
    zeroJoyY = joyY;
}

void update() {
    Wire.requestFrom (0x52, 6); // request data from nunchuck
    while (Wire.available ()) {
        // receive byte as an integer
        status[cnt] = _nunchuk_decode_byte (Wire.receive()); //
        cnt++;
    }
    if (cnt > 5) {
        lastZ = buttonZ;
        lastC = buttonC;
        lastJoyX = readJoyX();
        lastJoyY = readJoyY();
        //averageCounter++;
        //if (averageCounter >= AVERAGE_N)
        //    averageCounter = 0;
        cnt = 0;
        joyX = (status[0]);
        joyY = (status[1]);
        for (i = 0; i < 3; i++)
            //accelArray[i][averageCounter] = ((int)status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2)) >> ((i+1)*2));
            angles[i] = (status[i+2] << 2) + ((status[5] & (B00000011 << ((i+1)*2)) >> ((i+1)*2));
            //accelArray[averageCounter] = ((int)status[3] << 2) + ((status[5] & B00110000) >> 4);
            //accelZArray[averageCounter] = ((int)status[4] << 2) + ((status[5] & B11000000) >> 6);
            buttonZ = !(status[5] & B00000001);
            buttonC = !(status[5] & B00000010) >> 1;
            _send_zero(); // send the request for next bytes
    }
}

// UNCOMMENT FOR DEBUGGING
//byte * getStatus() {
//    // return status; //}
}

float readAccelX() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[0] - ZEROY; }
float readAccelY() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[1] - ZEROY; }
float readAccelZ() {
    // total = 0; // accelArray[xyz][averageCounter] * FAST_WEIGHT;
    return (float)angles[2] - ZEROZ; }

boolean zPressed() {
    return (buttonZ && !lastZ); }

boolean cPressed() {
    return (buttonC && !lastC); }

// for using the joystick like a directional button
boolean rightJoy(int thresh=60) {
    return (readJoyX() > thresh and lastJoyX <= thresh); }

// for using the joystick like a directional button
boolean leftJoy(int thresh=60) {
    return (readJoyX() < -thresh and lastJoyX >= -thresh); }

int readJoyX() {
    return (int)joyX - zeroJoyX; }

int readJoyY() {
    return (int)joyY - zeroJoyY; }

// R, the radius, generally hovers around 210 (at least it does with mine)
// int R() {
//    // return sqrt(readAccelX() * readAccelX() + readAccelY() * readAccelY() + readAccelZ() * readAccelZ());
//    }

// returns roll degrees
int readRoll() {
    return (int)(atan2(readAccelX(),readAccelZ()) / M_PI * 180.0); }

// returns pitch in degrees
int readPitch() {
    return (int)(acos(readAccelY()/RADIUS)/ M_PI * 180.0); // optionally swap 'RADIUS' for 'R()'

}

private:
    byte _nunchuk_decode_byte (byte x)
    {
        //decode is only necessary with certain initializations
        //x = (x ^ 0x17) + 0x17;
        return x; }

    void _send_zero()
    {
        Wire.beginTransmission (0x52); // transmit to device 0x52
        Wire.send (0x00); // sends one byte
        Wire.endTransmission (); // stop transmitting
    }
}; #endif
```

## Sketch PROCESSING:

```
/*
 * Wii controlled
 * RGB Cube.
 * The three primary colors of the additive color model are red, green, and blue.
 * This RGB color cube displays smooth transitions between these colors.
 */

float xmag, ymag = 0;
float newXmag, newYmag = 0;
int sensorCount = 5; // number of values to expect

import processing.serial.*;
Serial myPort; // The serial port
int BAUDRATE = 115200;
char DELIM = ','; // the delimeter for parsing incoming data

void setup()
{
size(200, 200, P3D);
noStroke();
colorMode(RGB, 1);
// Affiche la liste des ports série
println(Serial.list());
//myPort = new Serial(this, Serial.list()[0], BAUDRATE);
//affecte la bonne valeur au tableau Serial.list
myPort = new Serial(this, Serial.list()[1], BAUDRATE);
// clear the serial buffer:
myPort.clear();
}
float x, z;

void draw()
{
background(0.5, 0.5, 0.45);
pushMatrix();
translate(width/2, height/2, -30);
newXmag = mouseX/float(width) * TWO_PI;
newYmag = mouseY/float(height) * TWO_PI;
float diff = xmag-newXmag;
if (abs(diff) > 0.01) {
xmag -= diff/4.0;
}

diff = ymag-newYmag;
if (abs(diff) > 0.01) {
ymag -= diff/4.0;
}
}
```

```
// if ((sensorValues[1] > 15) && (sensorValues[1] < 165)) {
z = sensorValues[0] / 180 * PI;
x = sensorValues[1] / 180 * PI;
// }

rotateZ(z);
rotateX(x);
scale(50);
beginShape(QUADS);

fill(0, 1, 1);
vertex(-1, 1, 1);
fill(1, 1, 1);
vertex(1, 1, 1);
fill(1, 0, 1);
vertex(1, -1, 1);
fill(0, 0, 1);
vertex(-1, -1, 1);

fill(1, 1, 1);
vertex(1, 1, 1);
fill(1, 0, 0);
vertex(1, 1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 1);
vertex(1, -1, 1);

fill(1, 1, 0);
vertex(1, 1, -1);
fill(0, 1, 0);
vertex(-1, 1, -1);
fill(0, 0, 0);
vertex(-1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);

fill(0, 1, 0);
vertex(-1, 1, -1);
fill(0, 1, 1);
vertex(-1, 1, 1);
fill(0, 0, 1);
vertex(-1, -1, 1);
fill(0, 0, 0);
vertex(-1, -1, -1);

fill(0, 1, 0);
vertex(-1, 1, -1);
fill(1, 1, 0);
vertex(1, 1, -1);
fill(1, 1, 1);
vertex(1, 1, 1);
fill(0, 1, 1);
vertex(-1, 1, 1);

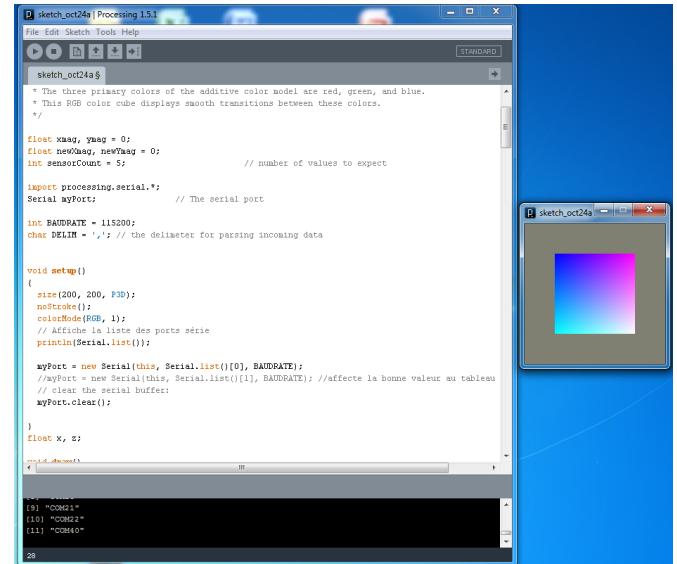
fill(0, 0, 0);
vertex(-1, -1, -1);
fill(1, 0, 0);
vertex(1, -1, -1);
fill(1, 0, 1);
vertex(1, -1, 1);
fill(0, 0, 1);
vertex(-1, -1, 1);
endShape();

popMatrix();
}

float[] sensorValues = new float[sensorCount]; // array to hold the incoming values

void serialEvent(Serial myPort) {
// read incoming data until you get a newline:
String serialString = myPort.readStringUntil('\n');
// if the read data is a real string, parse it:

if (serialString != null) {
//println(serialString);
//println(serialString.charAt(serialString.length()-3));
//println(serialString.charAt(serialString.length()-2));
// split it into substrings on the DELIM character:
String[] numbers = split(serialString, DELIM);
// convert each substring into an int
if (numbers.length == sensorCount) {
for (int i = 0; i < numbers.length; i++) {
// make sure you're only reading as many numbers as
// you can fit in the array:
if (i <= sensorCount) {
// trim off any whitespace from the substring:
numbers[i] = trim(numbers[i]);
sensorValues[i] = float(numbers[i]);
}
}
// Things we don't handle in particular can get output to the text window
print(serialString);
}
}
}
```

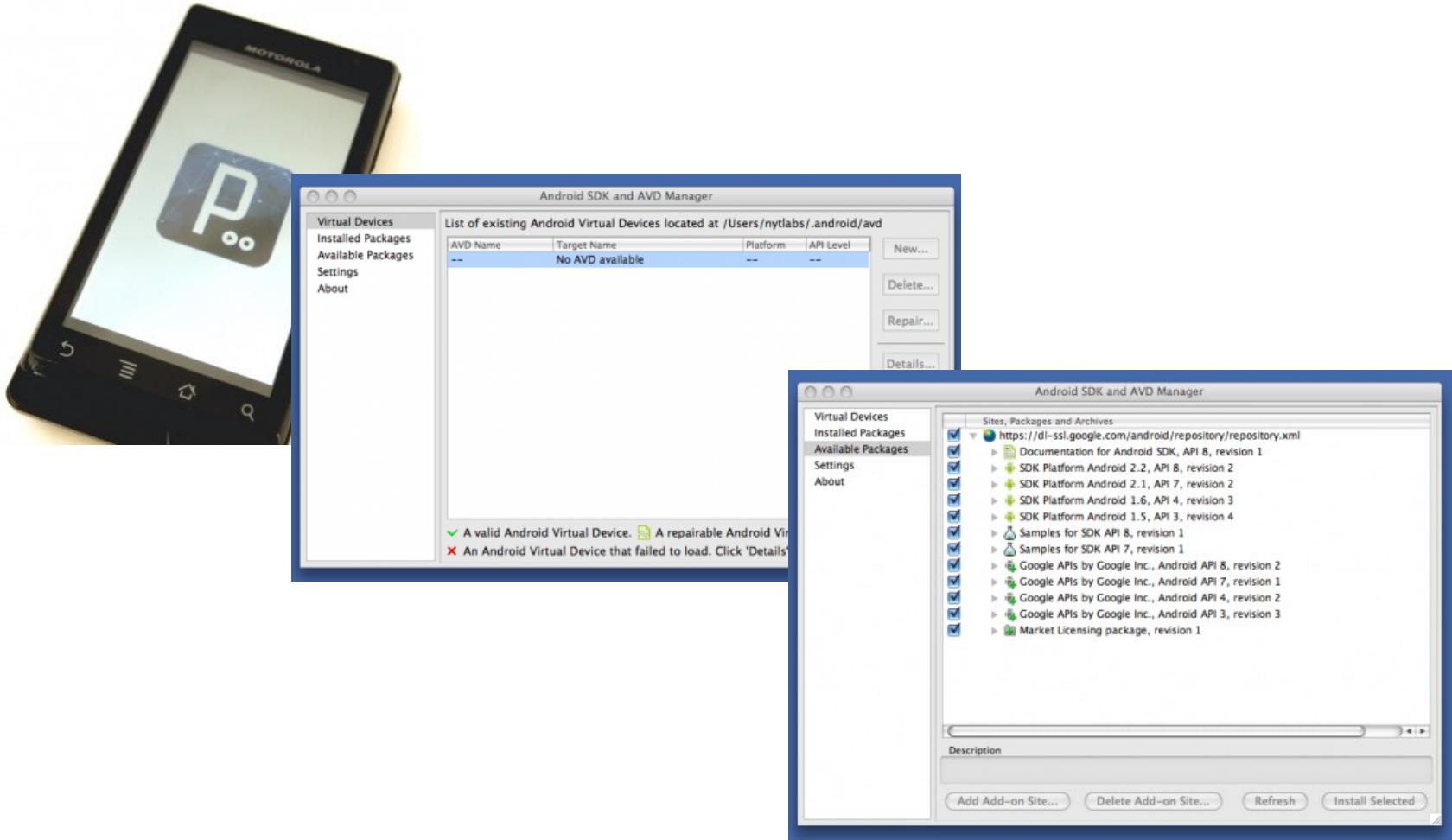


<http://arduino.cc/playground/Main/WiiChuckClass>

## IV-2 POUR ALLER PLUS LOIN: PROCESSING FOR ANDROID

# PROCESSING FOR ANDROID

Vous pouvez maintenant utiliser PROCESSING > V2.0 pour créer rapidement des applications pour les appareils mobiles qui utilisent le système d'exploitation Androïd.



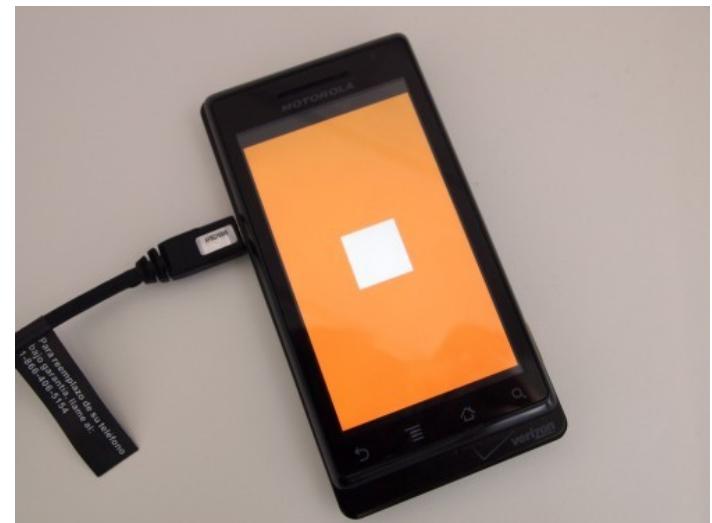
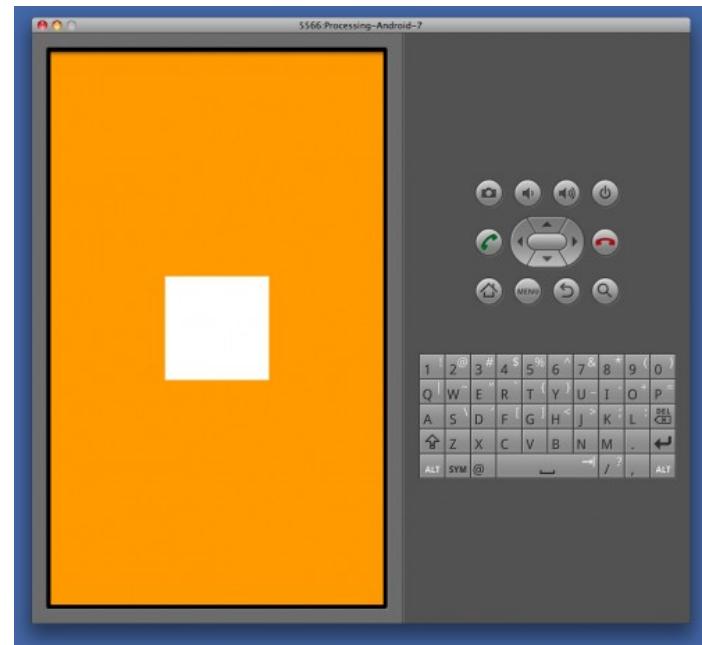
Note: vous n'avez pas besoin d'avoir un appareil Androïd pour faire ces applications, grâce à l'utilisation de l'émulateur logiciel.

<http://processing.org/tutorials/android/>

The screenshot shows the Processing IDE interface with the title bar "DroidTest | Processing 0190". The code window contains the following sketch:

```
void setup() {  
    size(480,800);  
    background(0xFF9900);  
    smooth();  
    noStroke();  
    fill(255);  
    rectMode(CENTER);  
}  
  
void draw() {  
    rect(width/2, height/2, 150, 150);  
}
```

The status bar at the bottom displays "Done Saving." and the number "13".



/\* World's simplest Android App!  
[blprnt@blprnt.com](mailto:blprnt@blprnt.com) Sept 25, 2010 \*/

```
//Build a container to hold the current rotation of the box
float boxRotation = 0;

void setup() {
    //Set the size of the screen (this is not really necessary in Android mode, but we'll do it anyway)
    size(480,800);
    //Turn on smoothing to make everything pretty.
    smooth();
    //Set the fill and stroke colour for the box and circle
    fill(255);
    stroke(255);
    //Tell the rectangles to draw from the center point (the default is the TL corner)
    rectMode(CENTER);
}

void draw() {
    //Set the background colour, which gets more red
    //as we move our finger down the screen.
    background(mouseY * (255.0/800), 100, 0);
    //Change our box rotation depending on how our finger has moved right-to-left
    boxRotation += (float)(pmouseX - mouseX)/100;

    //Draw the ball-and-stick
    line(width/2, height/2, mouseX, mouseY);
    ellipse(mouseX, mouseY, 40, 40);

    //Draw the box
    pushMatrix();
        translate(width/2, height/2);
        rotate(boxRotation);
        rect(0,0, 150, 150);
    popMatrix();
}
```



The screenshot shows the MIT App Inventor interface running in a Firefox browser window. On the left, the 'Premier\_Test' screen is displayed, featuring a list of components and their properties. Components include Screen1, ListPicker1, Button1-4, Label1-4, TextBox1-2, Clock1, BluetoothClient1, SpeechRecognizer1, and AccelerometerSensor1. The right side shows the 'App Inventor for Android Blocks Editor' where a script for 'Premier\_Test - Screen1' is being built. The script uses various blocks from the 'Built-In' palette (Text, Lists, Math, Logic, Control, Colors) to handle events like button clicks, list picker changes, and sensor data, and interact with a Bluetooth connection.

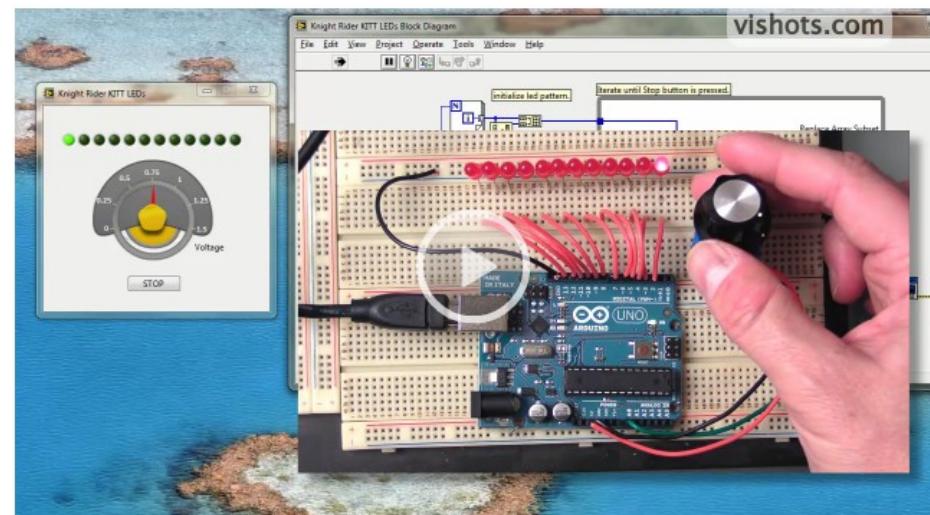
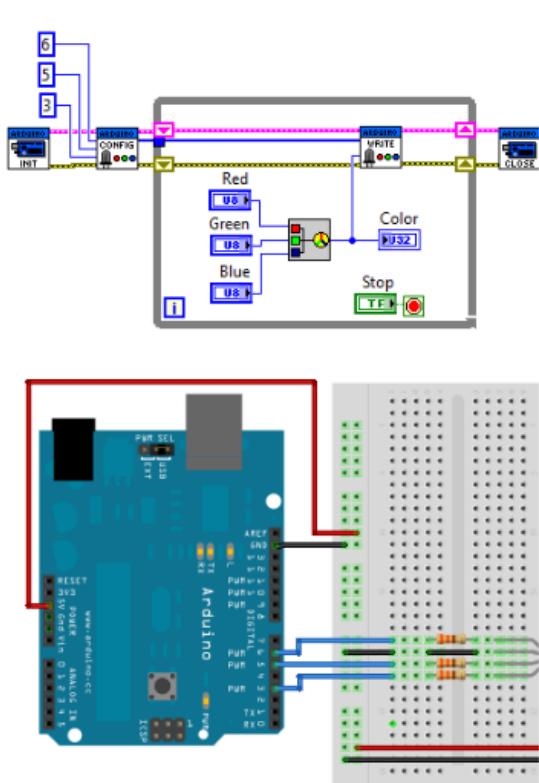
<http://beta.appinventor.mit.edu/>

<http://appinventor.mit.edu/>

RETOUR

## IV-2 POUR ALLER PLUS LOIN: LIFA

L'interface LabVIEW pour Arduino (LIFA) Toolkit permet aux développeurs d'acquérir des données à partir du microcontrôleur Arduino et le traiter dans l'environnement de programmation graphique LabVIEW et c'est gratuit !!! ;-)



### Top 5 des raisons qui vous rend plus productif avec LabVIEW lorsque vous utilisez Arduino:

- 1 - Interagissez avec votre système via une interface utilisateur graphique.
- 2 - Optimisez votre processus de conception avec la programmation graphique intuitive.
- 3 - Améliorer votre expérience de débogage avec des outils interactifs.
- 4 - Fonctions de mise en œuvre simple pour des tâches complexes.
- 5 - API ouverte permet une personnalisation complète - personnaliser vos programmes en fonction de votre application.

<https://decibel.ni.com/content/groups/labview-interface-for-arduino>  
<http://vishots.com/getting-started-with-the-labview-interface-for-arduino/>  
<http://innovelectronique.fr/2012/05/23/aduino-et-lifa-episode-2/>  
<http://innovelectronique.fr/2012/05/04/aduino-et-lifa-labview-interface-for-arduino/>

<https://decibel.ni.com/content/groups/labview-interface-for-arduino?view=overview#/?tagSet=undefined>

The screenshot shows a Microsoft Internet Explorer browser window displaying the National Instruments (NI) Community website for the "LabVIEW Interface for Arduino" group. The URL in the address bar is <https://decibel.ni.com/content/groups/labview-interface-for-arduino?view=overview#/?tagSet=undefined>. The page title is "LabVIEW Interface for Arduino".

The main content area displays a list of documents under the heading "Documents à la une". There are two items listed:

- LabVIEW Interface for Arduino Setup Procedure (par Sammy\_K)
- LabVIEW Interface for Arduino FAQ (par Sammy\_K)

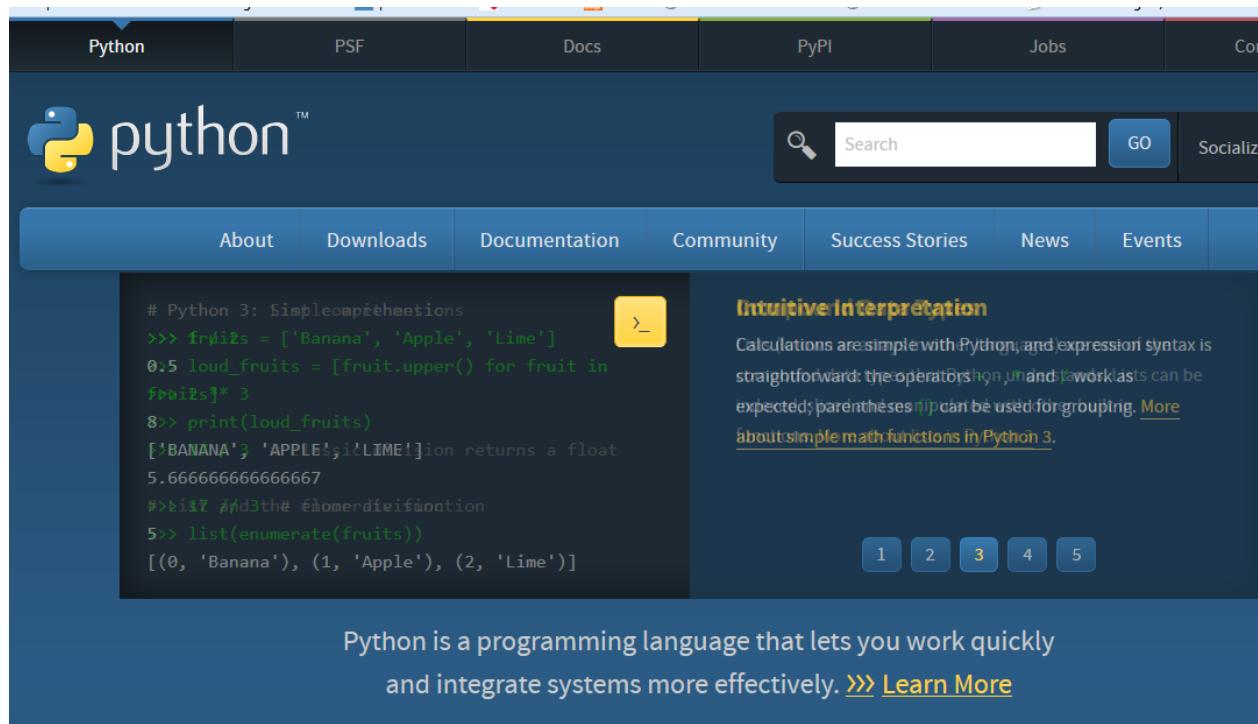
Below this, there is a larger list of "Documents" filtered by "Tous les documents". The list includes the following entries:

Auteur	Sujet	Visualiser	Classement	Dernier modifié
Peter_R	Arduino Example: Simple LED	83	★★★★★	29 déc. 2011 10:30 par Peter_R
rgodin	ITG3200	634	★★★★★	23 août 2011 07:21 par rgodin
JasonD	Ball and Paddle Game	1 120	★★★★★	8 août 2011 10:11 par JasonD
Sammy_K	LabVIEW Interface for Arduino Packets	2 083	★★★★★	27 juil. 2011 17:33 par Sammy_K
Nick_425	Arduino Push-Button: WAV Audio Sampler	1 783	★★★★★	12 juil. 2011 13:49 par Nick_425
Sammy_K	LabVIEW Interface for Arduino FAQ	5 056	★★★★★	9 juil. 2011 13:42 par Sammy_K
Sammy_K	LabVIEW Interface for Arduino Setup Procedure	17 538	★★★★★	9 juil. 2011 12:37 par Sammy_K
Sammy_K	Arduino BlinkM Example	1 715	★★★★★	23 mai 2011 14:20 par Sammy_K
Le_Sauvage	Ready to get started?	6 387	★★★★★	17 mai 2011 15:21

On the right side of the page, there are two sidebar sections: "Actions" and "Documents populaires". The "Actions" section contains links for "Avis" and "Fils du groupe". The "Documents populaires" section lists the same documents as the main list. Below these is a "Meilleurs participants" section showing user profiles for Sammy\_K, BenJ, Kevin\_F, MeghanK, hrh1818, and RymamW.

## IV-3 POUR ALLER PLUS LOIN: PYTHON

<https://www.python.org/>



The screenshot shows the Python.org homepage. At the top, there's a navigation bar with tabs for Python, PSF, Docs, PyPI, Jobs, and Community. Below the header is the Python logo and a search bar. The main content area features a code snippet in a dark box:

```
# Python 3: Simple arithmetic
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
...fruits]* 3
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']ion returns a float
5.666666666666667
#this add the #comment if you want
5>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

To the right of the code, there's a section titled "Intuitive Interpretation" with text about Python's syntax. Below the code snippet are five numbered buttons (1-5). Further down, a large call-to-action button says "Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)".

## Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

[Start with our Beginner's Guide](#)

## Download

Python source code and installers are available for download for all versions! Not sure which version to use? [Check here](#).

Latest: Python 3.5.2 - Python 2.7.12

## Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

[docs.python.org](#)

## Jobs

Looking for work or have a related position that you're hire for? Our [relaunched community-run job board](#) place to go.

[jobs.python.org](#)

## CODE: GESTION DE L'ENCODEUR ROTATOIRE ET ENVOIE DE LA CHAINE DE CARACTERE PAR ARDUINO VIA SON PORT SERIE:

Fichier Édition Croquis Outils Aide

Test\_Encoder\_Rotatoire\_LongPress\_Debounce\_PourProgramme\_Python | Arduino 1.6.9

```
#include <Arduino.h>
#include <SoftwareSerial.h>

#define PIN_ENC1 2
#define PIN_ENC2 3
#define PIN_EBTN 4
#define PIN_LED1 13

static boolean moving=false;
volatile int encValue = 0;
int encValue_max = 3;
int encValue_min = 0;
unsigned int encValueTracking = 1;
boolean enc1 = false;
boolean enc2 = false;

// Here I'm messing around with button press durations - we could go to town here!
//enum pressDuration {shortPress, normalPress, longPress, veryLongPress };
//long presses[] = {150, 300, 800, 1400 };
//char* pressText[]={"short press", "normal press", "long press", "very loooooooooong press"};

enum pressDuration {normalPress, longPress, veryLongPress };
long presses[] = {300, 800, 1400 };
char* pressText[]={"normal press", "long press", "very loooooooooong press"};
//#define DEBUG

void setup()
{
    pinMode(PIN_ENC1, INPUT);
    pinMode(PIN_ENC2, INPUT);
    pinMode(PIN_EBTN, INPUT);
    pinMode(PIN_LED1, OUTPUT);
    digitalWrite(PIN_ENC1, HIGH);
    digitalWrite(PIN_ENC2, HIGH);
    digitalWrite(PIN_EBTN, HIGH);
    Serial.begin(9600);
    //menuIntro();
    attachInterrupt(0, intrEncChange1, CHANGE);
    attachInterrupt(1, intrEncChange2, CHANGE);
}

void intrEncChange1()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC1) == enc1)
        return;
    enc1 = !enc1;
    if(enc1 == !enc2)
        encValue += 1;
}

void intrEncChange2()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC2) == enc2)
        return;
    enc2 = !enc2;
    if(enc2 == !enc1)
        encValue -= 1;
}

void loop()
{
    static unsigned long btnHeld = 0;
    moving = true;
    if(encValueTracking != encValue){
        //Serial.print("encValue: ");
        if (encValue < encValue_min){
            encValue = encValue_max;
        }else {
            if (encValue > encValue_max){
                encValue = encValue_min;
            }
        }
        //fin du else

        Serial.println(encValue);
        //Serial.println(encValue, DEC);
        //Serial.println(encValue, BIN);
        encValueTracking = encValue;
    }
}
```



Fichier Édition Croquis Outils Aide

Test\_Encoder\_Rotatoire\_LongPress\_Debounce\_PourProgramme\_Python | Arduino 1.6.9

```
moving = false;

}

void intrEncChange2()
{
    if(moving)
        delay(1);
    if(digitalRead(PIN_ENC2) == enc2)
        return;
    enc2 = !enc2;
    if(enc2 == !enc1)
        encValue -= 1;

    moving = false;

}

void loop()
{
    static unsigned long btnHeld = 0;
    moving = true;
    if(encValueTracking != encValue){
        //Serial.print("encValue: ");
        if (encValue < encValue_min){
            encValue = encValue_max;
        }else {
            if (encValue > encValue_max){
                encValue = encValue_min;
            }
        }
        //fin du else

        Serial.println(encValue);
        //Serial.println(encValue, DEC);
        //Serial.println(encValue, BIN);
        encValueTracking = encValue;
    }
}
```

Fichier Édition Croquis Outils Aide

Test\_Encoder\_Rotatoire\_LongPress\_Debounce\_PourProgramme\_Python | Arduino 1.6.9

```
// Upon button press...
if(digitalRead(PIN_EBTN) == LOW) && !btnHeld{
    btnHeld = millis();
    digitalWrite(PIN_LED1, HIGH);
#endif DEBUG
    Serial.print("pressed selecting: ");
    Serial.println(encValue);

}

// Upon button release...
if(digitalRead(PIN_EBTN) == HIGH) && btnHeld{
    long t = millis();
    t -= btnHeld;
    digitalWrite(PIN_LED1, LOW);
    int dur = veryLongPress;
    for(int i = 0; i<= veryLongPress; i++){
        if(t > presses[i])
            continue;
        dur = i;
        delay(100); //delay pour debounce
        break;
    }

#endif DEBUG
    Serial.print("released selecting: ");
    Serial.print(encValue, DEC);
    Serial.print(" (after ");
    Serial.print(t, DEC);
    Serial.print(" ms = ");
    Serial.print(pressText[dur]);
    Serial.println(")");
}

btnHeld = 0;
encValue=encValue_min; //reset de la valeur du compteur
//Serial.println(encValue);

}

void menuIntro()
{
    Serial.println("Encoder Menu - blah, blah, blah");
}
```

## CODE PYTHON DE LECTURE DU PORT SERIE SUR PC:

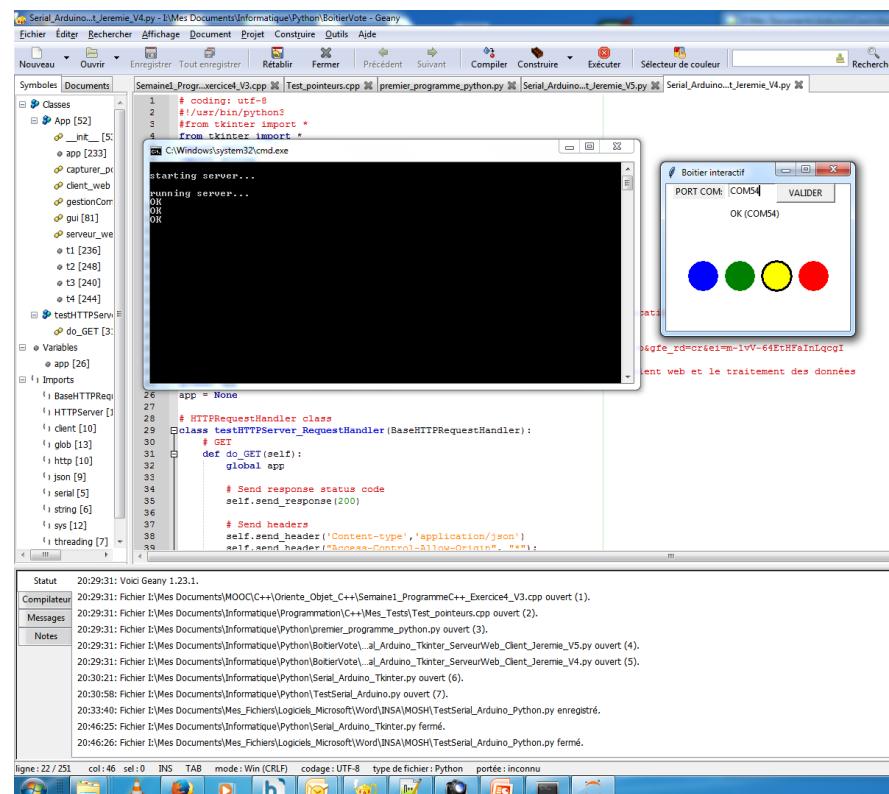
```

1 import serial
2 import string
3
4 ser = serial.Serial("COM54", 9600, timeout=1)
5 print (ser)
6
7 while 1:
8     donnee = str (ser.readline ())
9     tab = donnee.split ("') # Je découpe la chaîne en utilisant l'apostrophe (' ').
10    #print (tab)
11    if len (tab) == 3: # Ca doit retourner 3 éléments: [0]:"selected:b", [1]:"XX\r\n", et [2]:"".
12        v = tab [1] # On récupère le deuxième élément.
13        v = v.strip ("\r\n") # On enlève les retours chariots.
14        #print (v)
15    try:
16        if v.find("selected:") == 0:
17            v = v.replace('selected:', "")
18            #print ("La valeur sélectionnée est:", int(v))
19            print ("La valeur sélectionnée est:", int (v))
20        else :
21            v = int (v) # On convertit la valeur en numérique.
22    except:
23        v = None # Si ça échoue, on met NULL (None en Python).
24    if v is not None: # Si v a bien été trouvé
25        #print ("La valeur lue est:", v)
26        print (v)
27    else:
28        print ("ERREUR !")

```

## CODE PYTHON:

- LANCEMENT DU SERVEUR WEB
- CLIENT WEB
- INTERFACE GRAPHIQUE TKINTER



## CODE AJAX DE RECUPERATION DES DONNEES ENVOYEEES EN JSON A PARTIR DU SCRIPT PYHTON:

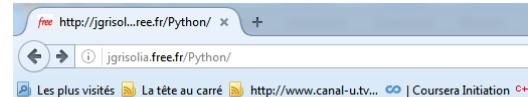
### Techniques AJAX – XMLHttpRequest

#### Envoi de données au format JSON

```

44 <script>
45 function maj()
46 {
47     var xhr = new XMLHttpRequest();
48     xhr.onreadystatechange = function() {
49         if (xhr.readyState == 4)
50         {
51             if (xhr.status == 200)
52             {
53                 /*
54                  readyState :
55                      Holds the status of the XMLHttpRequest. Changes from 0 to 4:
56                      0: request not initialized
57                      1: server connection established
58                      2: request received
59                      3: processing request
60                      4: request finished and response is ready
61                  status :
62                      200: "OK"
63                      404: Page not found
64                  */
65                 //console.log (xhr.responseText);
66                 var data = JSON.parse(xhr.responseText);
67                 document.getElementById("valeur").innerHTML = data.valeur;
68                 document.getElementById("select").innerHTML = data.select;
69                 /*document.getElementById("time").innerHTML = data.time;*/
70
71             /*MISE A JOUR DES CARRÉS DE COULEURS*/
72             for (var i = 0; i < 4; i++)
73             {
74                 var oid = "val" + i
75                 var obj = document.getElementById(oid)
76                 if (obj != null)
77                 {
78                     if (i == data.valeur)
79                     {
80                         obj.style.border = "5px solid"
81                         obj.style.borderColor = "black"
82                     }
83                     else
84                     {
85                         obj.style.border = "5px solid"
86                         obj.style.borderColor = "transparent"
87                     }
88                 }
89             }
90             setTimeout(maj, 100);
91         }
92     };
93     //xhr.open("GET", "http://localhost:8081", true); //envoi des données du local host vers
94     xhr.open("GET", "getval.php", true); //envoi des données de getval.php vers le serveur dis
95     xhr.send();
96
97 };
</script>
```

## PAGE WEB D'AFFICHAGE DES DONNEES AVEC MISE A JOUR DYNAMIQUE (~100ms)



AFFICHEZ LE CLASSEMENT

AFFICHEZ LES VOTES

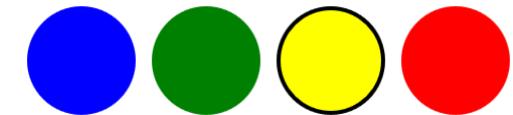
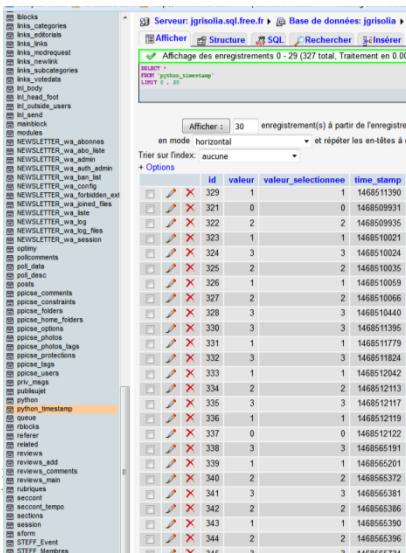
Valeur  
en cours

Valeur  
sélectionnée

2

0

## BDD DE STOCKAGE DES DONNEES



Nombre total de votes			
Valeur 0	Valeur 1	Valeur 2	Valeur 3
4	18	17	25



## PAGE WEB D'AFFICHAGE DYNAMIQUE DES DONNEES DE VOTES:

**IV-4 D'AUTRES PISTES  
INTERESSANTES ?**

Quand Microchip se réveille? Il fait des 32bits compatibles avec l'Arduino (8bits) et au même prix !!!!



**chipKIT Uno32™ Arduino™-Compatible Prototyping Platform**  
**\$26.95** [Add to Cart](#)

**Hide Details**

IC:	Microchip® PIC32MX320F128
Programming:	Application development using an environment based on the original Arduino™ IDE modified to support PIC32 that also still supports the original Arduino™ line. Leverages <a href="#">existing code examples, tutorials and resources</a> .

To download the IDE, please visit:  
<https://github.com/chipKIT32/chipKIT32-MAX/downloads>.

- Microchip® PIC32MX320F128 processor
  - 80 MHz 32-bit MIPS
  - 128K Flash, 16K SRAM
- Compatible with existing Arduino™ code examples, reference materials and other resources
- Can also be programmed using Microchip's MPLAB® IDE (along with a PICKit 3 and our PICKit3 Programming Cable Kit, seen below)
- Arduino™ "Uno" form factor
- Compatible with Arduino™ shields
- 42 available I/O
- User LED
- Connects to a PC using a USB A-> mini B cable (not included)

The chipKIT™ Uno32™ combines compatibility with the popular Arduino™ open source hardware prototyping platform with the performance of the Microchip PIC32 microcontroller. The Uno32 is the same form factor as the Arduino™ Uno board and is compatible with Arduino™ shields. It features a USB serial port interface for connection to the IDE and can be powered via USB or an external power supply.

The Uno32 board takes advantage of the powerful PIC32MX320F128 microcontroller. This microcontroller features a 32-bit MIPS processor core running at 80MHz, 128K of flash program memory and 16K of SRAM data memory.

The Uno32 can be programmed using an environment based on the original Arduino™ IDE modified to support PIC32. In addition, the Uno32 is fully compatible with the advanced Microchip MPLAB® IDE and the PICKit3 in-system programmer/debugger.

For additional platform-specific support for your chipKIT, please visit:  
<http://www.chipkit.org/forum/>.



**chipKIT Max32™ Arduino™-Compatible Prototyping Platform**  
**\$49.50** [Add to Cart](#)

**Hide Details**

IC:	Microchip® PIC32MX795F512
Programming:	Application development using an environment based on the original Arduino™ IDE modified to support PIC32 that also still supports the original Arduino™ line. Leverages <a href="#">existing code examples, tutorials and resources</a> .

To download the IDE, please visit:  
<https://github.com/chipKIT32/chipKIT32-MAX/downloads>.

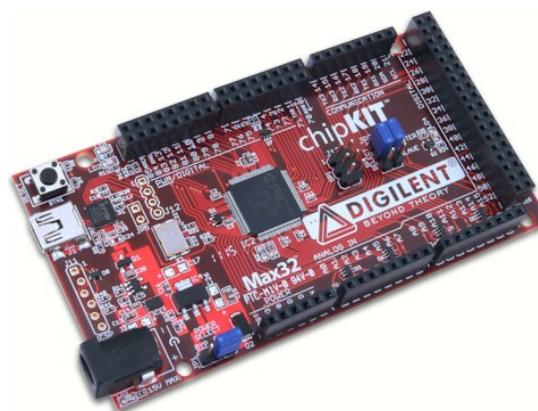
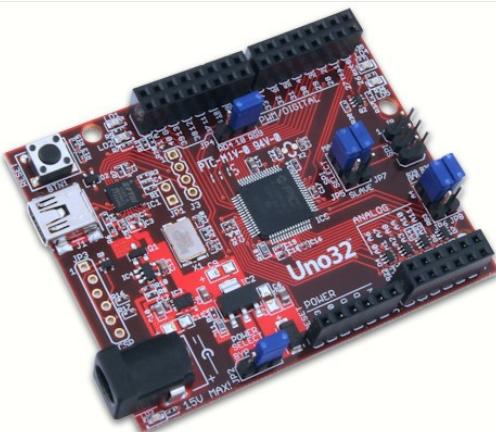
- Microchip® PIC32MX795F512 processor
  - 80 MHz 32-bit MIPS
  - 512K Flash, 128K RAM
  - USB 2.0 OTG controller
  - 10/100 Ethernet MAC
  - Dual CAN controllers
- Provides additional memory and advanced communications peripherals
- Compatible with existing Arduino code examples, reference materials and other resources
- Can also be programmed using Microchip's MPLAB® IDE (along with a PICKit 3 and our PICKit3 Programming Cable Kit, seen below)
- Arduino™ "Mega" form factor
- Compatible with Arduino™ shields
- 83 available I/O
- User LED
- Connects to a PC using a USB A-> mini B cable (not included)

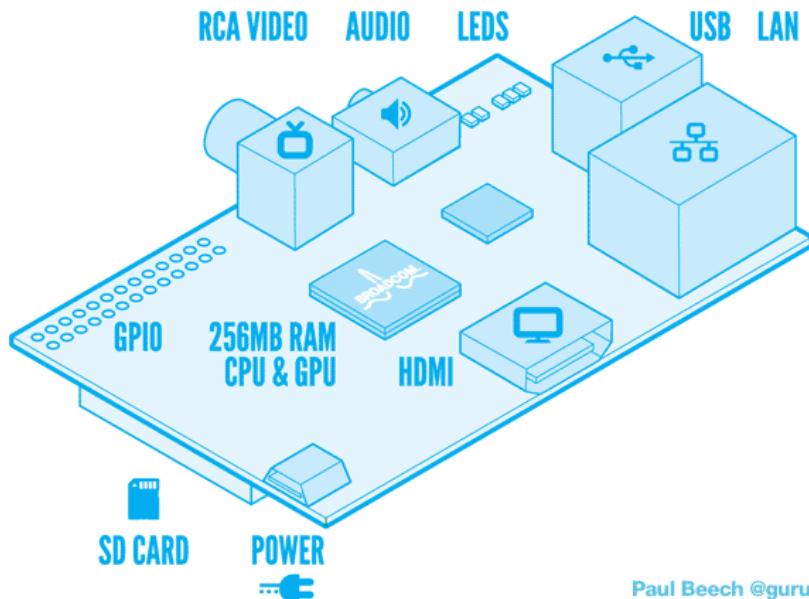
The chipKIT™ Max32™ combines compatibility with the popular Arduino™ open source hardware prototyping platform with the performance of the Microchip PIC32 microcontroller. The Max32 is the same form factor as the Arduino Mega board and is compatible with standard Arduino™ shields as well as larger shields for use with the Mega boards. It features a USB serial port interface for connection to the IDE and can be powered via USB or an external power supply.

The Max32 board takes advantage of the powerful PIC32MX795F512 microcontroller. This microcontroller features a 32-bit MIPS processor core running at 80MHz, 512K of flash program memory and 128K of SRAM data memory. In addition, the processor provides a USB 2 OTG controller, 10/100 Ethernet MAC and dual CAN controllers that can be accessed via add-on I/O shields.

The Max32 can be programmed using an environment based on the original Arduino™ IDE modified to support PIC32. In addition, the Max32 is fully compatible with the advanced Microchip MPLAB® IDE and the PICKit3 in-system programmer/debugger.

For additional platform-specific support for your chipKIT, please visit:  
<http://www.chipkit.org/forum/>.



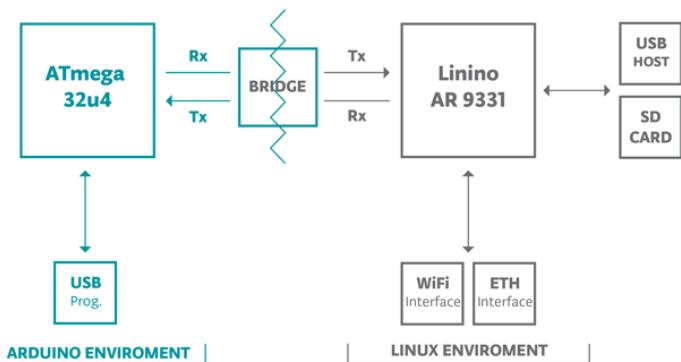
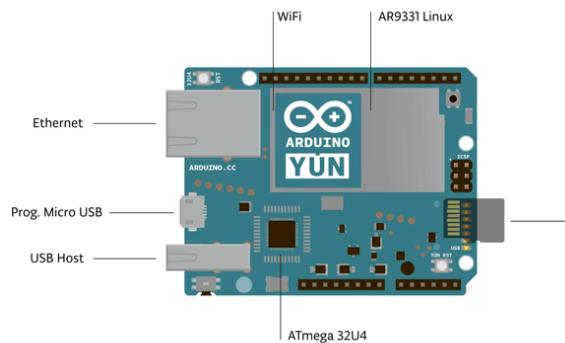


Paul Beech @guru

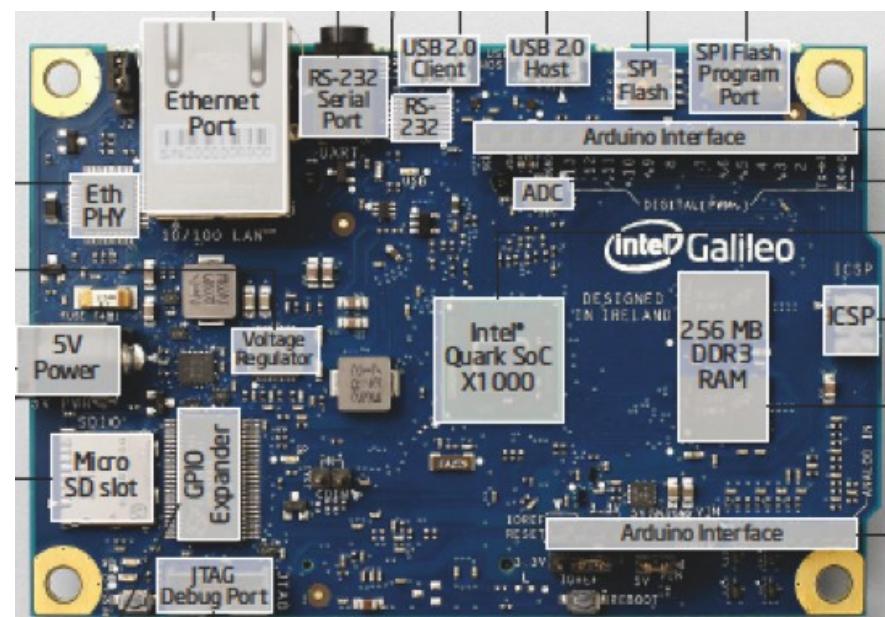
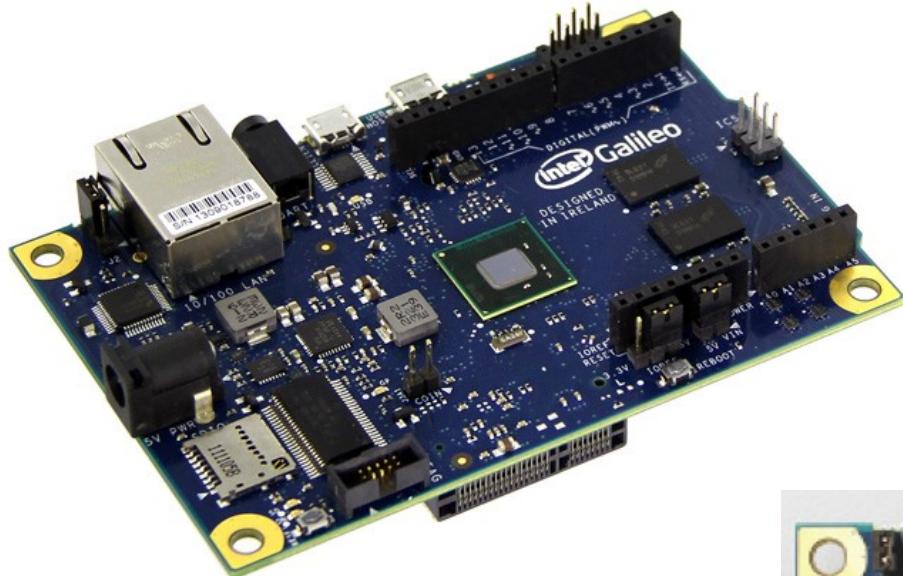


An ARM GNU/Linux box for \$25. Take a byte!

Mais attention à l'arrivée de l'ARDUINO YUN:



# INTEL GALILEO





## Technical specs

Microcontroller	Intel Curie
Operating Voltage	3.3V (5V tolerant I/O)
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 4 provide PWM output)
PWM Digital I/O Pins	4
Analog Input Pins	6
DC Current per I/O Pin	4 mA
Flash Memory	196 kB
SRAM	24 kB
Clock Speed	32MHz
Features	Bluetooth LE, 6-axis accelerometer/gyro
Length	68.6 mm
Width	53.4 mm

## Overview

3.3V    32-bit    32 MHz    ARC Core

A learning and development board that delivers the performance and low-power consumption of the Intel® Curie™ Module with the simplicity of Arduino at an entry-level price.

It keeps the same robust form factor and peripheral list of the UNO with the addition of onboard Bluetooth LE capabilities and a 6-axis accelerometer/gyro to help you easily expand your creativity into the connected world.

The module contains two tiny cores, an x86 (Quark) and an ARC, both clocked at 32Mhz. The Quark core runs ViperOS RTOS and helps the Arduino core to accomplish the most demanding tasks. It comes with 14 digital input/output pins (of which 4 can be used as PWM outputs), 6 analog inputs, a USB connector for serial communication and sketch upload, a power jack, an ICSP header with SPI signals and I2C dedicated pins.

The board operating voltage and I/O is 3.3V but all pins are protected against 5V overvoltage.

The Arduino 101 (USA only) and the Genuino 101 (outside USA) has been designed in collaboration with Intel®.

## V - LES REFERENCES INDISPENSABLES



**Démarrez avec Arduino: Principes de base et premiers montages, Massimo Banzi [français, 128 pages]**

Editions Dunod, 2011, ISBN-13: 978-2100562916

[chez l'éditeur](#)

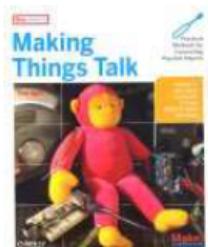


**Arduino - Maîtrisez sa programmation et ses cartes d'interface (shields), Christian Tavernier [français, 232 pages]**

Editions Dunod, 2011, ISBN-13: 978-2100559275

[Chez l'éditeur](#)

Site de l'auteur : <http://www.tavernier-c.com/>



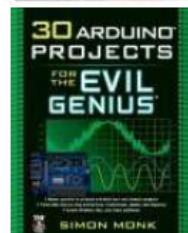
**Making Things Talk, Tom Igoe [anglais, 426 pages]**

Editions O'Reilly, 2007, ISBN-13 : 978-0-596-51051-0

Site de l'auteur : <http://www.tigoe.net/pcomp/>

Chez l'éditeur : <http://oreilly.com/catalog/9780596510510/>

Seconde édition prévue pour août 2011 : <http://oreilly.com/catalog/063692001092>



**30 Arduino Projects for the Evil Genius, Simon Monk [anglais, 208 pages]**

Editions McGraw Hill, 2010, ISBN : 978-0071741330

Chez l'éditeur : <http://www.mcgraw-hill.co.uk/html/007174133X.html>

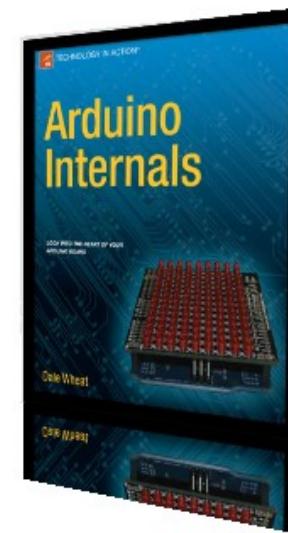
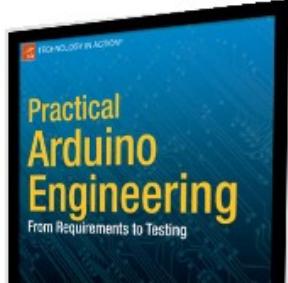
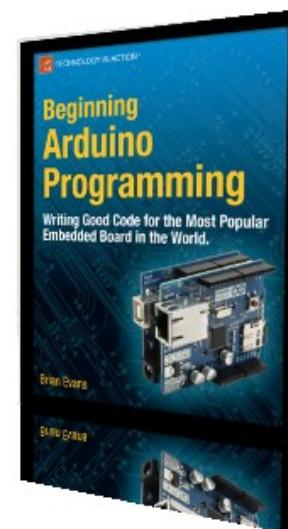
Site : <http://www.arduinoevilgenius.com/>



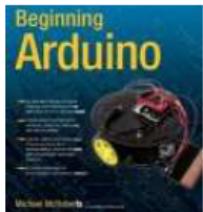
**Make: Arduino Bots and Gadgets, Learning by Discovery, Tero Karvinen & Kimmo Karvinen [anglais, 296 pages]**

Editions O'Reilly, 2011, ISBN-13: 978-1449389710

Chez l'éditeur : <http://oreilly.com/catalog/0636920010371/>



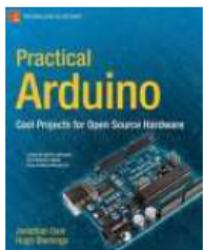
# LIVRES DE REFERENCES



## Beginning Arduino, Michael McRoberts [anglais, 472 pages]

Editions Apress, 2011, ISBN13: 978-1-4302-3240-7

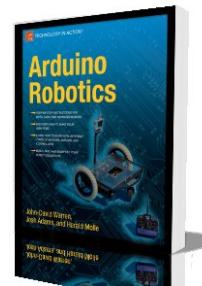
Chez l'éditeur : <http://www.apress.com/9781430232407>  
Code source : [http://www.apress.com/downloadable/down...le\\_id/358/](http://www.apress.com/downloadable/down...le_id/358/)



## Practical Arduino, Jonathan Oxer & Hugh Blemings [anglais, 456 pages]

Editions Apress, 2010, ISBN13: 978-1-4302-2477-8

Chez l'éditeur : <http://www.apress.com/9781430224778>  
Site : <http://www.practicalarduino.com/>  
Twitter : <http://twitter.com/parduino>



- ISBN13: 978-1-4302-3183-7
- 628 Pages
- User Level: Intermediate to Advanced
- Publication Date: July 18, 2011



## Arduino, a Quick Start Guide, Maik Schmidt [anglais, 296 pages]

Editions The Pragmatic Programmers, 2011, ISBN: 978-1-93435-666-1

Sur le site de l'éditeur : <http://pragprog.com/titles/msard/arduino>  
code source : [http://pragprog.com/titles/msard/source\\_code](http://pragprog.com/titles/msard/source_code)



Editions O'Reilly, 2011, ISBN-13 : 978-0-596-80247-9

Chez l'éditeur : <http://oreilly.com/catalog/9780596802486>



## Programming Interactivity, Joshua Noble [anglais, 712 pages]

Editions O'Reilly, 2009, ISBN-13 : 978-0-596-15414-1

«Programming Interactivity» couvre processing, arduino et openFrameworks.  
<http://oreilly.com/catalog/9780596154158> (google preview)

blog : <http://programminginteractivity.com>



## Open Softwear [anglais, 104 pages]

A paraître, BlushingBoy Publishing, 2011, ISBN : 978-91-97-95540-9

«Open Softwear is a book about fashion and technology. More precisely it is a book about Arduino boards, conductive fabric, resistive thread, soft buttons, LEDs, and some other things.»  
site : <http://softwear.cc/>  
La version beta de 2009 est téléchargeable sur le site

## RESSOURCES PRINCIPALES (ARDUINO/PROCESSING/FRITZING) & OPEN SOURCE HARDWARE:

Site principal : <http://arduino.cc/>

Téléchargement : <http://arduino.cc/en/Main/Software>

Guide de référence du langage : <http://arduino.cc/en/Reference/HomePage>

Forum de discussion : <http://arduino.cc/forum/>

Le forum français : <http://www.arduino.cc/cgi-bin/yabb2/YaBB.pl?board=francais>

Blog : <http://arduino.cc/blog/>

Wiki / playground (incontournable) : <http://arduino.cc/playground/>

wikipedia (fr) <http://fr.wikipedia.org/wiki/Arduino>

Liste des Shields: <http://shieldlist.org/>

Education: <http://scuola.arduino.cc/>

Processing: <http://processing.org/>

Fritzing : <http://fritzing.org/> (Prototypage Dessin des cartes , circuit graphique, schématique ou électronique)

Open Source Hardware: <http://dangerousprototypes.com/2011/06/30/tutorial-arduino-and-the-spi-bus/>

Et quand cela ne marche PAS !!! : <http://arduino.cc/en/Guide/Troubleshooting>

## SUR LE WEB:

Référence arduino en français : [http://www.mon-club-elec.fr/pmwiki\\_refe...n.HomePage](http://www.mon-club-elec.fr/pmwiki_refe...n.HomePage)

Référence en français: <http://www.mon-club-elec.fr/> : site perso amateur sur lequel je mets en ligne mes développements en électronique numérique programmée, dans une optique ludique et expérimentale, "just for fun" !

Initiation à Arduino (2006) : <http://www.craslab.org/arduino/livretht...oCRAS.html> (ou en version pdf)

The World Famous Index of Arduino & Freeduino Knowledge : <http://www.freeduino.org/>

Tutoriels de Tronixstuff <http://tronixstuff.wordpress.com/tutorials/>

Tutoriels d'Adafruit: <http://www.adafruit.com/tutorials>

Ressources / tutoriaux pour arduino sur le site du labomedia : [http://wiki.labomedia.org/index.php/Lie...ux\\_Arduino](http://wiki.labomedia.org/index.php/Lie...ux_Arduino)

Scratch pour Arduino : <http://seaside.citilab.eu/scratch/arduino>

Liste de discussion des développeurs d'arduino : <http://arduino.cc/mailman/listinfo/deve...arduino.cc>

Documentaire sur Arduino: <http://vimeo.com/18539129>

**GENERALISTES:**

[http://www.bricolec.com/\[fr\]](http://www.bricolec.com/[fr]) Portail francophone de l'arduino : Robotique, Bidouillage et hacking matériel (DIY)!

[http://arts-numeriques.codedrops.net/-Microcontrolleur-arduino-\[fr\]](http://arts-numeriques.codedrops.net/-Microcontrolleur-arduino-[fr])

<http://www.ladyada.net/learn/arduino/index.html>

<http://itp.nyu.edu/physcomp/Tutorials/ArduinoBreadboard>

<http://todbot.com/blog/bionicarduino/> [en] It is an introduction to microcontroller programming and interfacing with the real world using the Arduino physical computing platform. It focuses on building new physical senses and making motion with the building blocks of robotics, using Arduino as a platform.

•Très bon tutoriels progressifs : <http://todbot.com/blog/spookyarduino/>

•le site de Tom Igoe <http://tigoe.net/pcomp/>

•les sites Make, Hackaday, Sensorwiki, We make money not art, et bien d'autres encore pour avoir des conseils, des schémas, des idées....

•les sites de service de liens, de vidéos et d'images ( google, delicious, youtube, flickr,)

<http://luckylarry.co.uk/arduino-projects/arduino-weblinks-projects/>

<http://hacknmod.com/hack/top-40-arduino-projects-of-the-web/>

<http://santy.wikidot.com/arduino>

<http://zedomax.com/blog/2010/02/16/top-10-diy-arduino-projects-and-how-to-tutorials/>

Liens divers: <http://www.uni-weimar.de/medien/wiki/Arduino/Links>

<http://barzilouik.free.fr/wiki/doku.php?id=arduino:tutorial>

**REVENDEURS:**

<http://shop.snootlab.com/> (Toulouse !!!)

<http://www.lextronic.fr/>

<http://www.robotshop.com/eu/arduino-en.html>

<http://boutique.semageek.com/fr/2-arduino>

<http://adafruit.com/>

<http://www.sparkfun.com/>

**COMMUNICATION EN GENERAL:** <http://arduino.cc/playground/Main/InterfacingWithHardware#Communication>

I<sup>2</sup>C: <http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html>

<http://www.atmicropog.com/cours/I2C/i2c.php>

<http://www.esacademy.com/en/library/technical-articles-and-documents/miscellaneous/i2c-bus.html>

[http://www.robot-electronics.co.uk/acatalog/I2C\\_Tutorial.html](http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html)

<http://www.neufeld.newton.ks.us/electronics/?p=241>

Librairie Wii-Nunchuk: <https://github.com/lgeek/Arduino-Wii-Nunchuk>

TOP 30 utilisation du Nunchuk: <http://hacknmod.com/hack/top-30-wiimote-hacks-of-the-web/>

<http://johnnylee.net/projects/wii/>

<http://www.planete-sciences.org/robot/ressources/electronique/protocoleI2C/index.html>

<http://www.pobot.org/Controle-avec-un-Wii-Nunchuck.html>

Wii Motion Plus + Nunchuck: <http://voidbot.net/nunchchuk-and-wii-motion-plus-6-DOF.html>

<http://randomhacksofboredom.blogspot.com/2009/06/wii-motion-plus-arduino-love.html>

<http://www.freelug.org/spip.php?article917>

<http://www.aurel32.net/elec/i2c.php>

<http://todbot.com/blog/bionicarduino/>

<http://todbot.com/blog/2010/09/25/softi2cmaster-add-i2c-to-any-arduino-pins/>

<http://tronixstuff.wordpress.com/2010/10/20/tutorial-arduino-and-the-i2c-bus/>

<http://tronixstuff.wordpress.com/2010/10/29/tutorial-arduino-and-the-i2c-bus-part-two/>

<http://tronixstuff.wordpress.com/2010/05/20/getting-started-with-arduino-%E2%80%93-chapter-seven/>

<http://www.nearfuturelaboratory.com/2009/07/17/lis302dl-a-3-axis-accelerometer/>

[http://wiire.org/Main\\_Page](http://wiire.org/Main_Page)

[http://wiire.org/Wii/protocols/wiimote\\_bus#Wiimote\\_Bus\\_Pins\\_.286-pin\\_proprietary\\_connector\\_on\\_Wiimote.29](http://wiire.org/Wii/protocols/wiimote_bus#Wiimote_Bus_Pins_.286-pin_proprietary_connector_on_Wiimote.29)

## Note:

There are both 7- and 8-bit versions of I<sup>2</sup>C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. If you have a datasheet or sample code that uses 8 bit address, you'll want to drop the low bit (i.e. shift the value one bit to the right), yielding an address between 0 and 127.

CAPTEURS: <http://itp.nyu.edu/physcomp/sensors/Reports/Reports>

vers l'interface MIDI avec une arduino: <http://itp.nyu.edu/physcomp/Labs/MIDIOutput>

RFID: <http://www.sparkfun.com/products/8419>

<http://jadiema.blogspot.com/2011/03/arduino-rfid-reader-piezo-sounder.html>

COMPASS: <http://lusorobotica.com/index.php/topic,36.0.html>, <http://wiring.org.co/learning/libraries/hmc6352sparkfun.html>

ROBOT: <http://arduino103.blogspot.com/2011/06/comment-dimensionner-un-moteur.html>

<http://www.pobot.org/> [fr] L'association POBOT a été créée en 2003, par un groupe de passionnés de robotique, dans le but : d'apprendre ensemble, de se perfectionner, d'échanger, de s'entre-aider.

<http://www.robot-maker.com/index.php?/topic/3380-divers-robots-a-a-base-darduino-et-de-wifi/>

<http://robot.aspi.free.fr/>

APPROCHE CLIENT-SERVEUR AVEC LIAISON WIFI : <http://www.mon-club-elec.fr/pmwiki> mon club elec/pmwiki.php?n=MAIN.ArduinoExpertSymbioseArduinoPCPrincipeDeploiement

NUNCHUK MEETS NXT: <http://www.mindstormsforum.de/viewtopic.php?t=3828>

ANDROID ADK MEETS ARDUINO: <http://www.amarino-toolkit.net/index.php/my-apps.html>

<http://www.mobot.es/MobotBTCar.html>

Vélo pour Geek: <http://www.michaelfretz.com/2011/01/03/punkt-fizen-android-and-arduino-powered-bike-navigation/>

<http://www.labradoc.com/i/follower/p/android-arduino-accessory>

<http://www.labradoc.com/i/follower>

<http://www.semageek.com/handbag-une-application-android-pour-piloter-ladk-arduino-sans-programmation/>

CELLULES SOLAIRES : <http://www.doctormonk.com/2011/09/arduino-solar-radio.html>

RASPBERRY-Pi : <http://www.doctormonk.com/2012/04/raspberry-pi-and-arduino.html>

POUR LES ARTISTES:

<http://softwear.cc/book/2011/open-softwear-2nd-edition/page/2/>

<http://web.media.mit.edu/~leah/LilyPad/>

TAPIS SENSITIF, ÉCRANS MULTI-TOUCH...: <http://numuscus.pascsaq.org/?p=551>

<http://www.interface-z.com/>

<http://codelab.fr/2552>, partie table Multi-Touch et reactive: <http://codelab.fr/1076>

Introduction au système Arduino :

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

Vue d'ensemble des cartes :

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

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

Définition de l'open source ! et plein d'autres bricoles => Lien avec les shields...

Le Logiciel :

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

La carte Duemilanove :

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

Le brochage :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.REFERENCESBrochageDuemilanove](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.REFERENCESBrochageDuemilanove)

Compilation :

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

Premier programme :

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

Références :

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

Références Etendues:

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

Les librairies de références :

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

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

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

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

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

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

<http://www.aurel32.net/elec/lcd.php> , [http://fabrice.sincere.pagesperso-orange.fr/cm\\_electronique/projet\\_pic/LCDAlpha/LCDAlpha.htm](http://fabrice.sincere.pagesperso-orange.fr/cm_electronique/projet_pic/LCDAlpha/LCDAlpha.htm)

Comment écrire une librairie: <http://www.robotix.fr/tutoriel-1-59-creer-une-bibliotheque-arduino.html>

[http://www.robot-maker.com/index.php?user/4963-Philippe-RX/page\\_tab\\_tutorials](http://www.robot-maker.com/index.php?user/4963-Philippe-RX/page_tab_tutorials)

Fiche sur LCD => Communication avec LCD : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD)

Fiche Touch Shield Nintendo DS (Entrée analogique et Sortie Digitale)

Fiche sur Nunchuck (accelerometer) => Bus I2C

Bus SPI <http://www.arduino.cc/playground/Code/Spi>

Pourquoi ne pas écrire dans une carte SD ? => il faut le shield Ethernet officiel:

<http://www.ladyada.net/learn/arduino/ethfiles.html>

Fiche sur l'etherennet Shield : <http://www.ladyada.net/learn/arduino/ethfiles.html>

Tester les interruptions externes => Optocoupleur à fourche : [http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheTestSimple](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoInitiationEntreesOnOffOptoFourcheTestSimple)

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.MaterielCapteurRoptofourcheLTH301-07](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.MaterielCapteurRoptofourcheLTH301-07)

<http://www.gotronic.fr/catalog/opts/interrupteurs.htm>

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.ArduinoExpertServoPanSuiviBalle](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertServoPanSuiviBalle)

Focus sur les Shields

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.MATERIEL](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.MATERIEL)

Alimentation de PC:

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.TECHNIQUETrucsUtiliserAlimPC](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TECHNIQUETrucsUtiliserAlimPC)

LIENS PROCESSING :

[http://www.mon-club-elec.fr/pmwiki\\_mon\\_club\\_elec/pmwiki.php?n=MAIN.OUTILSProcessing](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.OUTILSProcessing)

<http://www.scoop.it/t/arduino-processing>

ARDUINO MEETS PROCESSING:

[http://webzone.k3.mah.se/projects/arduino-workshop/projects/arduino\\_meets\\_processing/instructions/index.html](http://webzone.k3.mah.se/projects/arduino-workshop/projects/arduino_meets_processing/instructions/index.html)

ARDUINO MEETS FLASH:

<http://www.fabiobiondi.com/blog/2009/11/arduino-and-flash-modify-your-existent-flash-games-to-work-with-a-joystick/>

<http://www.fabiobiondi.com/blog/2009/10/arduino-and-flex-connect-a-joystick-to-your-application/>

Divers :

Générateur de code: [http://www.mon-club-elec.fr/pmwiki\\_generateur\\_code/pmwiki.php?n>Main.HomePage](http://www.mon-club-elec.fr/pmwiki_generateur_code/pmwiki.php?n>Main.HomePage)

R2D2 Translator: <http://www.r2d2translator.com/>

ARDUINO MEETS LABVIEW: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/209835>

<http://vishots.com/getting-started-with-the-labview-interface-for-arduino/>

Et un article pour réfléchir: <http://www.framablog.org/index.php/post/2010/06/02/arduino-materiel-libre>

# LIENS

## OPEN SOURCE HARDWARE ET LICENCE LIBRE

[http://en.wikipedia.org/wiki/Open\\_source\\_hardware](http://en.wikipedia.org/wiki/Open_source_hardware)

<http://fr.wikipedia.org/wiki/Copyleft>

[http://fr.wikipedia.org/wiki/Licence\\_Creative\\_Commons](http://fr.wikipedia.org/wiki/Licence_Creative_Commons)

<http://freedomdefined.org/Definition/Fr>

<http://creativecommons.org/licenses/by-nc-sa/2.0/fr/>

<http://opensource.org/osd>

## PWM

<http://www.siteduzero.com/sciences/tutoriels/arduino-pour-bien-commencer-en-electronique-et-en-programmation/transformation-pwm-signal-analogique>

<http://arduino-info.wikispaces.com/Arduino-PWM-Frequency>

<http://linuxedu.ac-toulouse.fr/doku.php?id=materiel:arduino:ressources:pedagogiques>

[http://www.planete-sciences.org/robot/boiteabots/index.php?option=com\\_content&task=view&id=44](http://www.planete-sciences.org/robot/boiteabots/index.php?option=com_content&task=view&id=44)

<http://planet.madeinfr.org/tag/arduino>

Le TETALAB: <http://tetalab.org/>

Le Tetalab est un hackerspace toulousain créé en juin 2009, avec un cadre décontracté pour des projets software/hardware.



## Structure

```
void setup() void loop()
```

## Control Structures

```
if (x<5){} else {}  
switch (myvar) {  
    case 1:  
        break;  
    case 2:  
        break;  
    default:  
}  
  
for (int i=0; i <= 255; i++){}  
while (x<5){}  
do {} while (x<5);  
continue; // Go to next in  
do/for/while loop  
return x; // Or 'return;' for voids.  
goto // considered harmful :-)
```

## Further Syntax

```
// (single line comment)  
/* (multi-line comment) */  
#define DOZEN 12 //Not baker's!  
#include <avr/pgmspace.h>
```

## General Operators

```
= (assignment operator)  
+ (addition) - (subtraction)  
* (multiplication) / (division)  
% (modulo)  
== (equal to) != (not equal to)  
< (less than) > (greater than)  
<= (less than or equal to)  
>= (greater than or equal to)  
&& (and) || (or) ! (not)
```

## Pointer Access

```
& reference operator  
* dereference operator
```

## Bitwise Operators

```
& (bitwise and) | (bitwise or)  
^ (bitwise xor) ~ (bitwise not)  
<< (bitshift left) >> (bitshift right)
```

## Compound Operators

```
++ (increment) -- (decrement)  
+= (compound addition)  
-= (compound subtraction)  
*= (compound multiplication)  
/= (compound division)  
&= (compound bitwise and)  
|= (compound bitwise or)
```

## Constants

```
HIGH | LOW  
INPUT | OUTPUT  
true | false  
143 // Decimal number  
0173 // Octal number  
0b11011111 //Binary  
0x7B // Hex number  
7U // Force unsigned  
10L // Force long  
15UL // Force long unsigned  
10.0 // Forces floating point  
2.4e5 // 240000
```

## Data Types

```
void  
boolean (0, 1, false, true)  
char (e.g. 'a'-128 to 127)  
unsigned char (0 to 255)  
byte (0 to 255)  
int (-32,768 to 32,767)  
unsigned int (0 to 65535)  
word (0 to 65500 (0 to 65535)  
long (-2,147,483,648 to  
2,147,483,647)  
unsigned long (0 to 4,294,967,295)  
float (-3.4028235E+38 to  
3.4028235E+38)  
double (currently same as float)  
sizeof(myint) // returns 2 bytes
```

## Strings

```
char S1[15];  
char S2[8]={'a','r','d','u','l','n','o'};  
char S3[8]={'a','r','d','u','l','n','o','\0'};  
//Included \0 null termination  
char S4[] = "arduino";  
char S5[8] = "arduino";  
char S6[15] = "arduino";
```

## Arrays

```
int myInts[6];  
int myPins[] = {2, 4, 8, 3, 6};  
int mySensVals[6] = {2, 4, -8, 3, 2};
```

## Conversion

```
char() byte()  
int() word()  
long() float()
```

## Qualifiers

```
static // persists between calls  
volatile // use RAM (nice for ISR)  
const // make read-only  
PROGMEM // use flash
```

## Digital I/O

```
pinMode(pin, [INPUT,OUTPUT])  
digitalWrite(pin, value)  
int digitalRead(pin)  
//Write High to inputs to use pull-up res
```

## Analog I/O

```
analogReference([DEFAULT,  
INTERNAL,EXTERNAL])  
int analogRead(pin) //Call twice if  
switching pins from high Z source.  
analogWrite(pin, value) // PWM
```

## Advanced I/O

```
tone(pin, freqhz)  
tone(pin, freqhz, duration_ms)  
noTone(pin)  
shiftOut(dataPin, clockPin,  
[MSBFIRST,LSBFIRST], value)  
unsigned long pulseIn(pin,[HIGH,LOW])
```

## Time

```
unsigned long millis() // 50 days overflow.  
unsigned long micros() // 70 min overflow  
delay(ms)  
delayMicroseconds(us)
```

## Math

```
min(x, y) max(x, y) abs(x)  
constrain(x, minval, maxval)  
map(val, fromL, fromH, toL, toH)  
pow(base, exponent) sqrt(x)  
sin(rad) cos(rad) tan(rad)
```

## Random Numbers

```
randomSeed(seed) // Long or int  
long random(max)  
long random(min, max)
```

## Bits and Bytes

```
lowByte()  
highByte()  
bitRead(x,bitn)  
bitWrite(x,bitn,bit)  
bitSet(x,bitn)  
bitClear(x,bitn)  
bit(bitn) //bitn: 0-LSB 7-MSB
```

## External Interrupts

```
attachInterrupt(interrupt, function,  
[LOW,CHANGE,RISING,FALLING])  
detachInterrupt(interrupt)  
interrupts()  
noInterrupts()
```

## Libraries:

### Serial.

```
begin(300, 1200, 2400, 4800,  
9600,14400, 19200, 28800, 38400,  
57600,115200))
```

```
end()
```

```
int available()
```

```
int read()
```

```
flush()
```

```
print()
```

```
println()
```

```
writeln()
```

### EEPROM (#include <EEPROM.h>)

```
byte read(intAddr)  
write(intAddr,myByte)
```

### Servo (#include <Servo.h>)

```
attach(pin , [min_uS, max_uS])  
write(angle) // 0-180  
writeMicroseconds(uS) //1000-  
2000,1500 is midpoint  
read() // 0-180  
attached() //Returns boolean  
detach()
```

### SoftwareSerial(RxPin,TxPin)

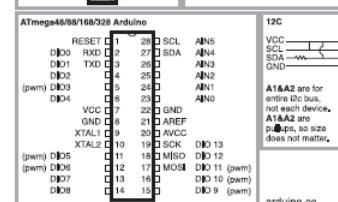
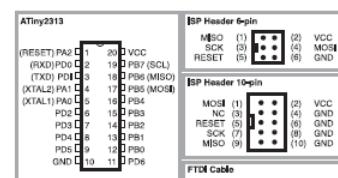
```
// #include<SoftwareSerial.h>  
begin(longSpeed) // up to 9600  
char read() // blocks till data  
print(myData) or println(myData)
```

### Wire (#include <Wire.h>) // For I2C

```
begin() // Join as master  
begin(addr) // Join as slave @ addr  
requestFrom(address, count)  
beginTransmission(addr) // Step 1  
send(mybyte) // Step 2  
send(char * mystring)  
send(byte * data, size)  
endTransmission() // Step 3  
byte available() // Num of bytes  
byte receive() // Return next byte  
onReceive(handler)  
onRequest(handler)
```

	ATMega168	ATMega328	ATMega1280
Flash (2k for bootloader)	16kB	32kB	128kB
SRAM	1kB	2kB	8kB
EEPROM	512B	1kB	4kB

	Duemillanove/ Nano/Pro/ ProMini	Mega
# of IO	14 + 6 analog (Nano has 14 + 8)	54 + 16 analog
Serial Pins	0 • RX 1 • TX	0 • RX1 1 • TX1 17 • RX2 18 • TX2 17 • RX3 16 • TX3 15 • RX4 14 • TX4
Ext. Interrupts	2 • (Int 0) 1 • (Int 1)	2,3,21,20,19,18 (IRQ0 - IRQ5)
PWM Pins	5,6 • Timer 0 9,10 • Timer 1 3,11 • Timer 2	0 - 13
SPI	10 • SS 11 • MOSI 12 • MISO 13 • SCK	53 - SS 51 - MOSI 50 - MISO 52 - SCK
I2C	Analog4 • SDA Analog5 • SCK	20 • SDA 21 - SCL



arduinocc

**ANALOG** : Analogique (0 à 1023 sur l'arduino) .

**AREF** : Abréviaison pour Analog REFérence, référence analogique.

**AVAILABLE** : Disponible.

**BEGIN** : Début.

**BIT** : bit, unité d'information informatique pouvant prendre soit la valeur 0 soit la valeur 1.

**BUFFER** : Tampon, dans le sens de "zone tampon". Mémoire temporaire

**BYTE** : Octet, soit un groupe de 8 bits.

**BPS** : Abréviaison pour Bits Per Second, Bits Par Seconde. Attention, abréviation toujours en minuscules.

**BREADBOARD**: plaque d'expérimentation

**CAPACITOR**: condensateur

**CHAR** : Pour CHARacter, caractère (typographique). Type de variable d'une taille d'un octet. C'est un synonyme de "byte" utilisé pour déclarer des variables stockant un caractère ou des chaînes de caractères.

**DEFINE** : Définit.

**DIGITAL** : Numérique (0 (LOW) ou 1 (HIGH))

**DO** : Faire.

**FALSE** : Faux.

**FOR** : Pour. Jusqu'à ce que.

**GND** : Abréviaison pour GrouND, la terre. C'est la masse, 0 Volt.

**HIGH** : Haut.

**ICSP**: Abréviaison pour In Circuit Serial Programming, programmation série sur circuit.

**IF / THEN / ELSE** : Si / Alors / Sinon.

**IN** : Souvent l'abréviation pour INput, Entrée. Est toujours en rapport avec le sens extérieur vers carte Arduino.

**INCLUDE** : Inclut.

**INPUT** : Entrée.

**IS** : Est (souvent dans le sens d'une question : Est ?).

**INT** : Abréviaison pour INTeger, entier. Groupe de 16 bits, 2 octets groupés, considérés comme représentant un nombre entier négatif ou positif.

**LONG** : Abréviaison pour "entier long". Groupe de 32 bits, 4 octets groupés, considérés comme représentant un nombre entier négatif ou positif.

**LOOP** : Boucle.

**LOW** : Bas.

**OUT** : Souvent l'abréviation pour OUTput, Sortie. Est toujours en rapport avec le sens carte Arduino vers extérieur.

**OUTPUT** : Sortie.

**PIN** : Broche d'E/S connectée à quelque chose (e.g. une LED).

**POWER** : Puissance, alimentation.

**PWM** : Abréviaison de (Pulse Width Modulation), soit Modulation en Largeur d'Impulsion.

**PWR** : Abréviaison pour PoWeR, puissance, alimentation.

**READ**: Lire.

**RESISTOR**: résistance.

**RELAY**: relais.

**RX** : Abréviaison pour Receive, réception.

**SERIAL** : Série.

**SETUP** : Initialisation.

**SENSOR**: capteur

**SKETCH**: programme qui tourne sur le micro-contrôleur

**SWITCH** : basculer, interrupteur

**TRUE** : Vrai.

**TX**: Abréviaison pour Transmit, transmission.

**WIRING**: câble, fils montrant les interconnexions physiques des composants.

**WHILE** : Tant que.

**WORD** : mot, soit dans le sens de langage ; soit dans le sens d'un groupe de 16 bits, 2 octets groupés considérés comme représentant un nombre entier positif ( $\geq 0$ ).

**WRITE**: Ecrire.