Arduino

Table des matières

1. Introduction....2
2. Le logiciel....5
  2.1. L'interface....5
  2.2. Le langage Arduino....7
3. Le matériel....9
  3.1. Constitution de la carte....9
  3.2. Test de la carte....10
4. Gestion des entrées / sorties....14
  4.1. La diode électroluminescente....14
  4.2. Temporisation....16
    4.2.1. Fonction delay()....16
    4.2.2. Fonction millis()....17
  4.3. Le bouton poussoir....19
  4.4. Les interruptions matérielles....21
  4.5. Afficheurs 7 segments....22
5. Communication par la liaison série....29
  5.1. Envoi de données....29
  5.2. Reception des données....30
6. Les grandeurs analogiques....32
  6.1. Les entrées analogiques....32
  6.2. Les sorties "analogiques"....35
7. Les écrans LCD....38
  7.1. Afficher du texte....40
  7.2. Créer un caractère....42
  7.3. Défilement de texte....42

Arduino est un circuit imprimé en matériel libre sur lequel se trouve un microcontrôleur qui peut être programmé pour analyser et produire des signaux électriques, de manière à effectuer des tâches très diverses comme la domotique (le contrôle des appareils domestiques - éclairage, chauffage…), le pilotage d'un robot, etc.

1. Introduction

Le système Arduino donne la possibilité d'allier les performances de la programmation à celles de l'électronique. Plus précisément, pour programmer des systèmes électroniques. Le gros avantage de l'électronique programmée c'est qu'elle simplifie grandement les schémas électroniques et par conséquent, le coût de la réalisation, mais aussi la charge de travail à la conception d'une carte électronique.

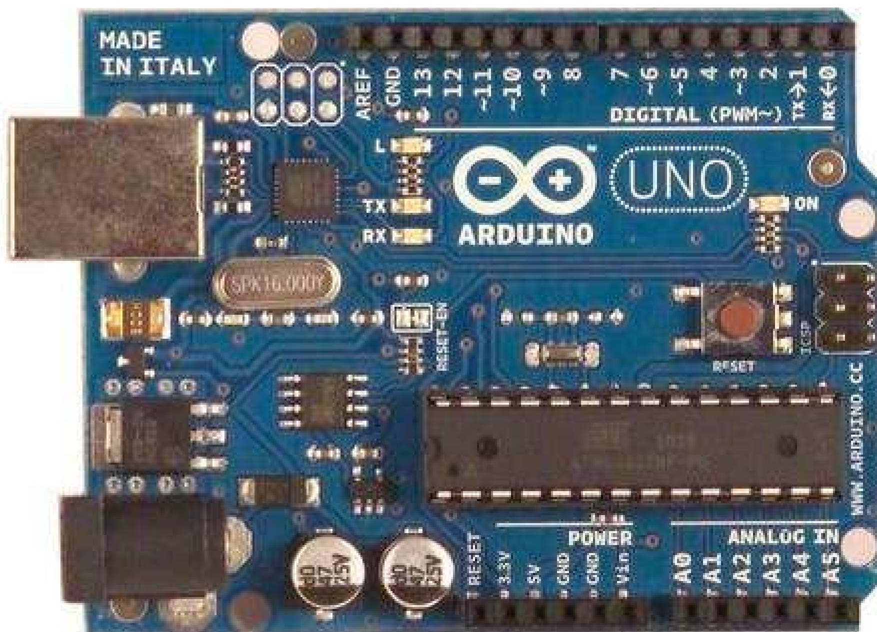
Le système Arduino permet de :

- contrôler les appareils domestiques
- fabriquer votre propre robot
- faire un jeu de lumières
- communiquer avec l'ordinateur
- télécommander un appareil mobile (modélisme)
- etc.

Le système Arduino est composé de deux choses principales : le matériel et le logiciel.

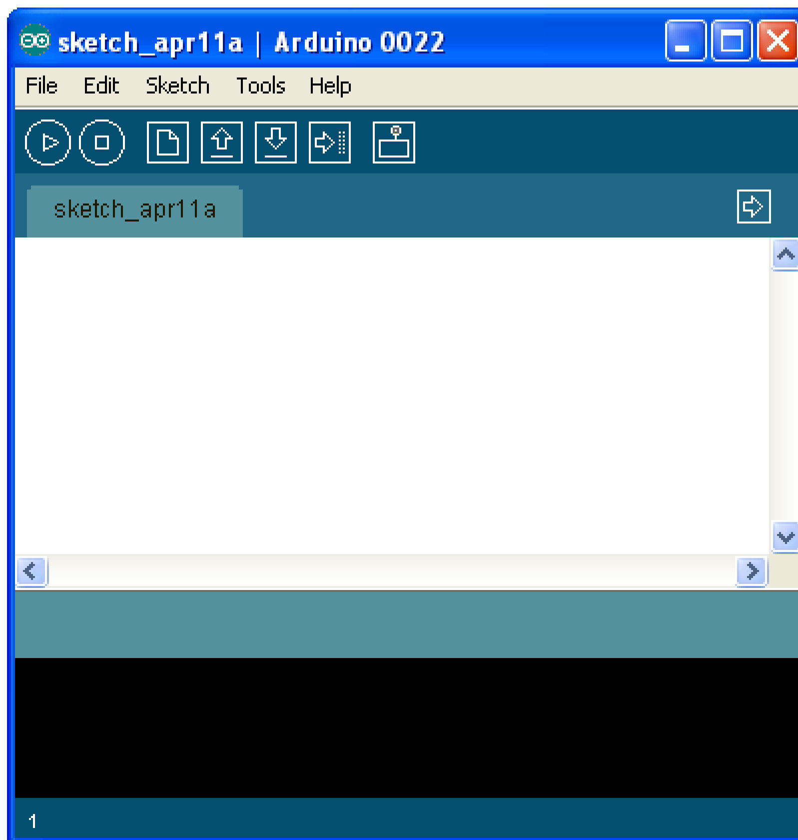
- Le matériel

  Il s'agit d'une carte électronique basée autour d'un microcontrôleur Atmega du fabricant Atmel, dont le prix est relativement bas pour l'étendue possible des applications.



- Le logiciel

  Le logiciel permet de programmer la carte Arduino. Il offre une multitude de fonctionnalités.

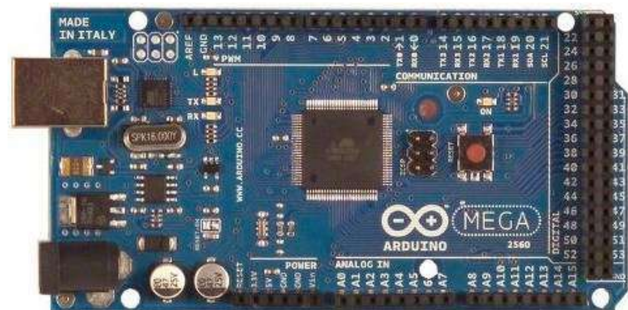


Il y a trois types de cartes :

- Les « officielles » qui sont fabriquées en Italie par le fabricant officiel : Smart Projects
- Les « compatibles » qui ne sont pas fabriqués par Smart Projects, mais qui sont totalement compatibles avec les Arduino officielles.
- Les « autres » fabriquées par diverse entreprise et commercialisées sous un nom différent (Freeduino, Seeduino, Femtoduino, ...).

carte Uno et Duemilanove

carte Mega



La carte Arduino est équippé d'un microcontrôleur. Le microcontrôleur est un composant

électronique programmable. On le programme par le biais d'un ordinateur grâce à un langage informatique, souvent propre au type de microcontrôleur utilisé.

Un microcontrôleur est constitué par un ensemble d'éléments qui ont chacun une fonction bien déterminée. Il est en fait constitué des mêmes éléments que sur la carte mère d'un ordinateur :

1. La mémoire

   Il en possède 5 types :

   - La **mémoire Flash** : C'est celle qui contiendra le programme à exécuter.Cette mémoire est effaçable et ré-inscriptible.
   - **RAM** : c'est la mémoire dite "vive", elle va contenir les variables de votre programme. Elle est dite "volatile" car elle s'efface si on coupe l'alimentation du micro-contrôleur.
   - **EEPROM** : C'est le disque dur du microcontrôleur. Vous pourrez y enregistrer des infos qui ont besoin de survivre dans le temps, même si la carte doit être arrêtée. Cette mémoire ne s'efface pas lorsque l'on éteint le microcontrôleur ou lorsqu'on le reprogramme.
   - Les **registres** : c'est un type de mémoire utilisé par le processeur.
   - La mémoire **cache** : c'est une mémoire qui fait la liaison entre les registres et la RAM.

2. Le processeur

   C'est le composant principal du micro-contrôleur. C'est lui qui va exécuter le programme qu'on lui donnerons à traiter. On le nomme souvent le CPU.

Pour que le microcontrôleur fonctionne, il lui faut une alimentation ! Cette alimentation se fait en générale par du +5V. D'autres ont besoin d'une tension plus faible, du +3,3V.

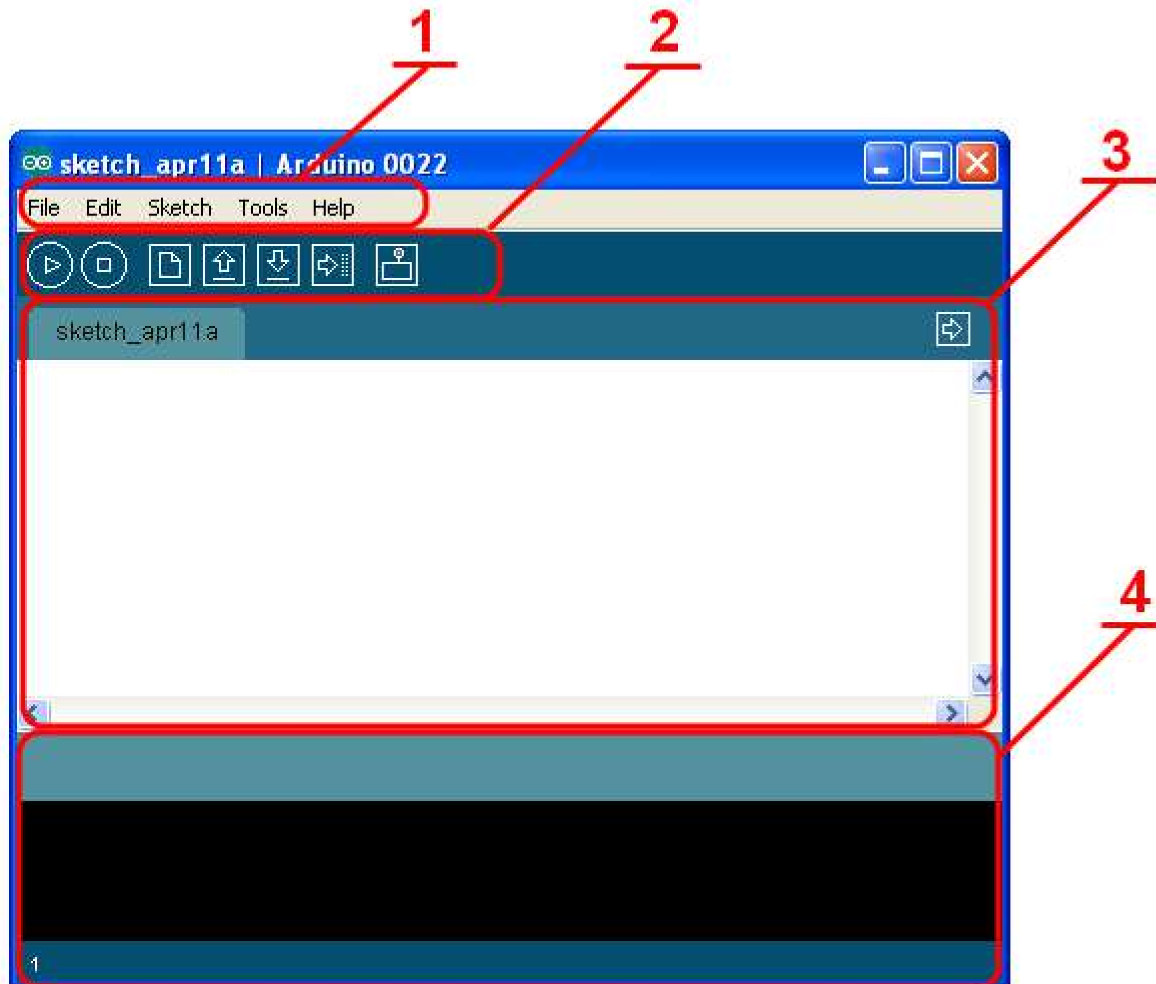
En plus d'une alimentation, il a besoin d'un signal d'horloge. C'est en fait une succession de 0 et de 1 ou plutôt une succession de tension 0V et 5V. Elle permet en outre de cadencer le fonctionnement du microcontrôleur à un rythme régulier. Grâce à elle, il peut introduire la notion de temps en programmation.

2. Le logiciel

Au jour d'aujourd'hui, l'électronique est de plus en plus remplacée par de l'électronique programmée. On parle aussi d'électronique embarquée ou d'informatique embarquée.

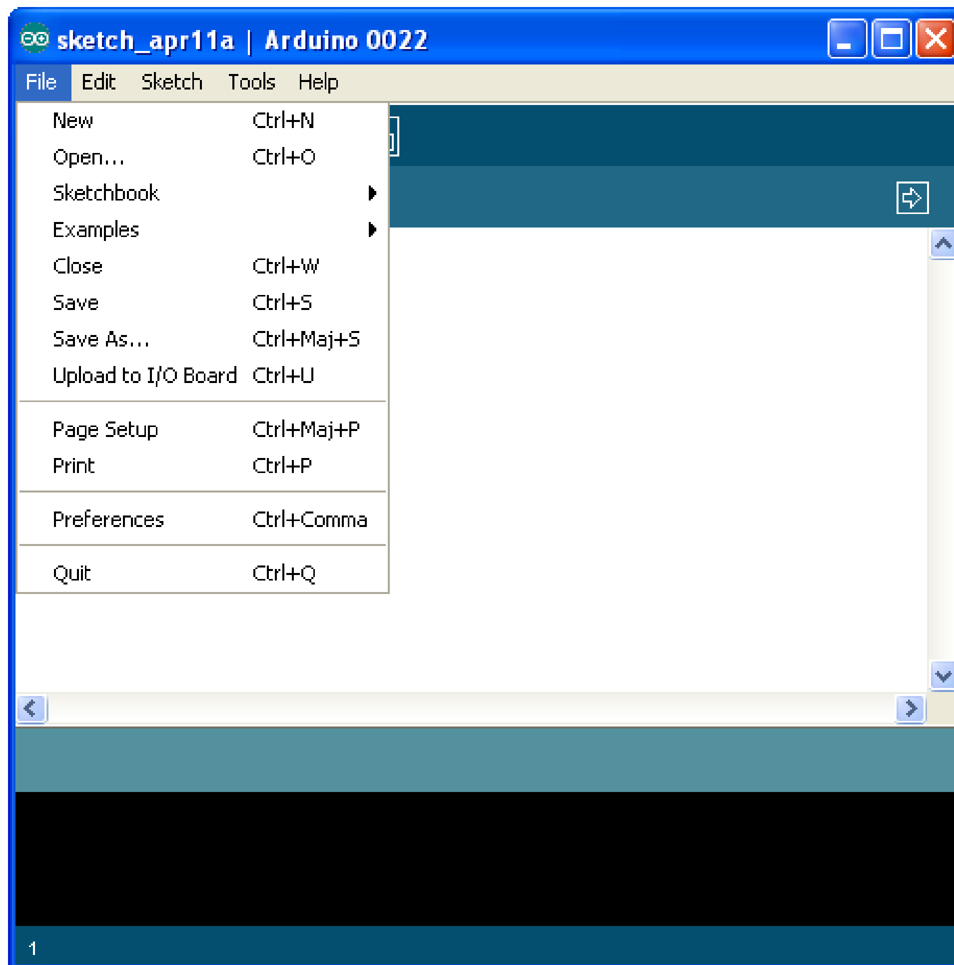
2.1. L'interface

L'interface du logiciel Arduino se présente de la façon suivante :



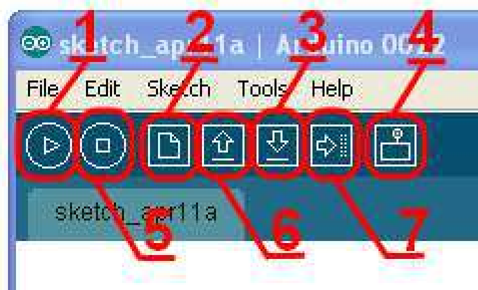
1. options de configuration du logiciel
2. boutons pout la programmation des cartes
3. programme à créer
4. débogueur (affichage des erreurs de programmation)

Le menu File dispose d’un certain nombre de choses qui vont être très utiles :



- New (nouveau) : va permettre de créer un nouveau programme. Quand on appuie sur ce bouton, une nouvelle fenêtre, identique à celle-ci, s'affiche à l'écran.
- Open... (ouvrir) : avec cette commande, on peut ouvrir un programme existant.
- Save / Save as... (enregistrer / enregistrer sous...) : enregistre le document en cours / demande où enregistrer le document en cours.
- Examples (exemples) : ceci est important, toute une liste se déroule pour afficher les noms d'exemples de programmes existant.

Les boutons



1. permet de vérifier le programme, il actionne un module qui cherche les erreurs dans le programme

2. Créer un nouveau fichier
3. Sauvegarder le programme en cours
4. Liaison série
5. Stoppe la vérification
6. Charger un programme existant
7. Compiler et envoyer le programme vers la carte

2.2. Le langage Arduino

Le projet Arduino était destiné à l'origine principalement à la programmation multimédia interactive en vue de spectacle ou d'animations artistiques. C'est une partie de l'explication de la descendance de son interface de programmation de Processing.

Processingest une librairie java et un environnement de développement libre. Le logiciel fonctionne sur Macintosh, Windows, Linux, BSD et Android.

Références :

- Le langage Java.
- Le langage C.
- L'algorithmique.

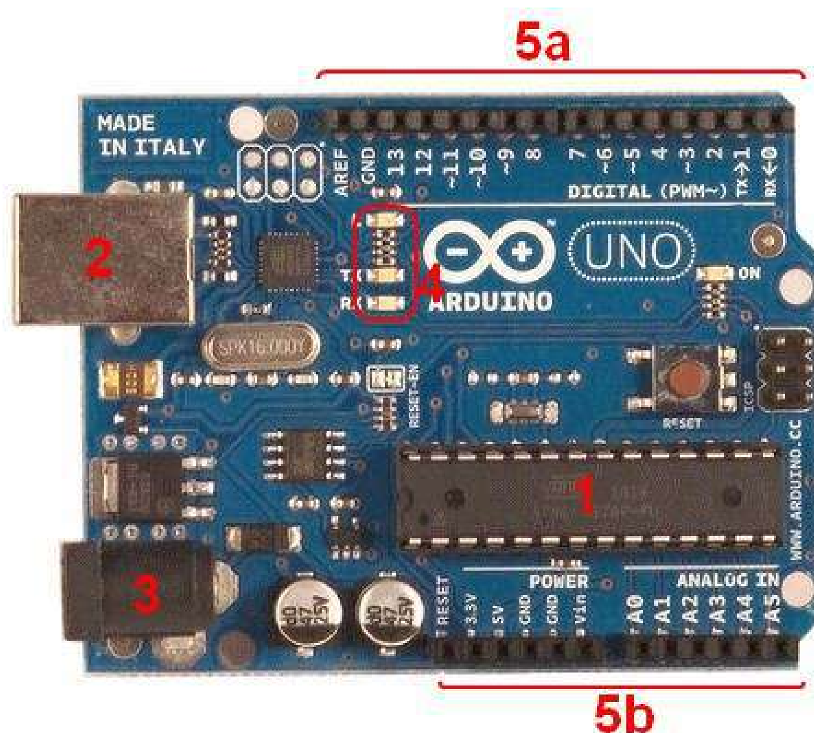
Cependant, le projet Arduino a développé des fonctions spécifiques à l'utilisation de la carte qui ont été listées ci-dessous. Vous obtiendrez la description de chacune d'elles dans le manuel de référence.

Structure	Constants	Functions
• setup() • loop()	• HIGH, LOW • INPUT, OUTPUT, INPUT_PULLUP • LED_BUILTIN	E/S numérique • pinMode() • digitalWrite() • digitalRead() E/S analogique • analogReference() • analogRead() • analogWrite() - PWM E/S avancée • tone() • noTone() • shiftOut() • shiftIn() • pulseIn() Temps • millis() • micros()

		• delay() • delayMicroseconds() Bits et octets • lowByte() • highByte() • bitRead() • bitWrite() • bitSet() • bitClear() • bit() Interruptions externes • attachInterrupt() • detachInterrupt() Interruptions • interrupts() • noInterrupts() Communication • Serial • Stream

3. Le matériel

3.1. Constitution de la carte



1. Le micro-contrôleur

Il va recevoir le programme et le stocker dans sa mémoire puis l'exécuter.

2, 3 : Alimentation

Pour fonctionner, la carte a besoin d'une alimentation. Le microcontrôleur fonctionnant sous 5V, la carte peut être alimentée en 5V par le port USB (en 2) ou bien par une alimentation externe (en 3) qui est comprise entre 7V et 12V. Cette tension doit être continue et peut par exemple être fournie par une pile 9V. Un régulateur se charge ensuite de réduire la tension à 5V pour le bon fonctionnement de la carte.

4. Visualisation

Les trois "points blancs" entourés en rouge sont des LED dont la taille est de l'ordre du millimètre. Ces LED servent à deux choses :

- Celle tout en haut du cadre : elle est connectée à une broche du microcontrôleur et va servir pour tester le matériel.

  Nota : Quand on branche la carte au PC, elle clignote quelques secondes.

- Les deux LED du bas du cadre : servent à visualiser l'activité sur la voie série (une pour l'émission et l'autre pour la réception). Le téléchargement du programme dans le micro-contrôleur se faisant par cette voie, on peut les voir clignoter lors du chargement.

5a, 5b : La connectique

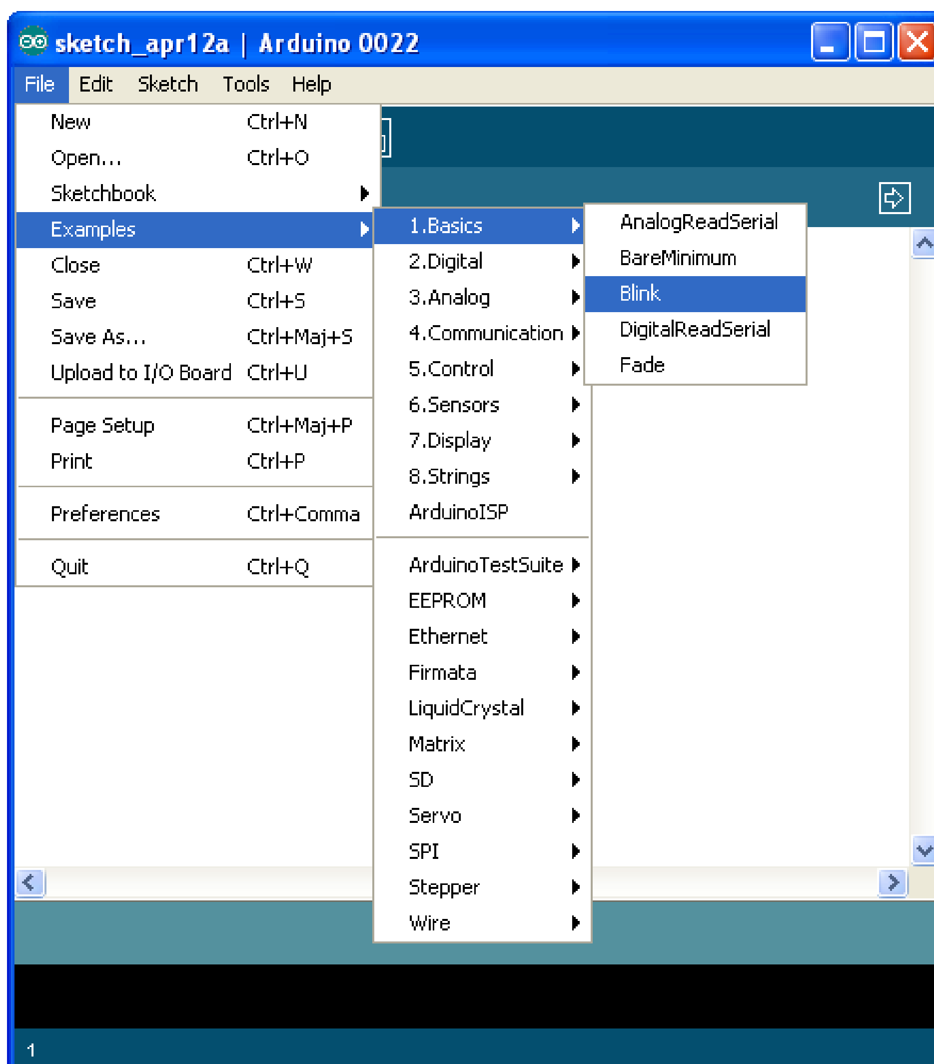
La carte Arduino ne possédant pas de composants qui peuvent être utilisés pour un programme, mis a par la LED connectée à la broche 13 du microcontrôleur, il est nécessaire de les rajouter. Mais pour ce faire, il faut les connecter à la carte (en 5a et 5b).

C'est grâce à cette connectique que la carte est "extensible", car l'on peut y brancher tous types de montages et modules ! Par exemple, la carte Arduino Uno peut être étendue avec des shields, comme le « Shield Ethernet » qui permet de connecter cette dernière à internet.

3.2. Test de la carte

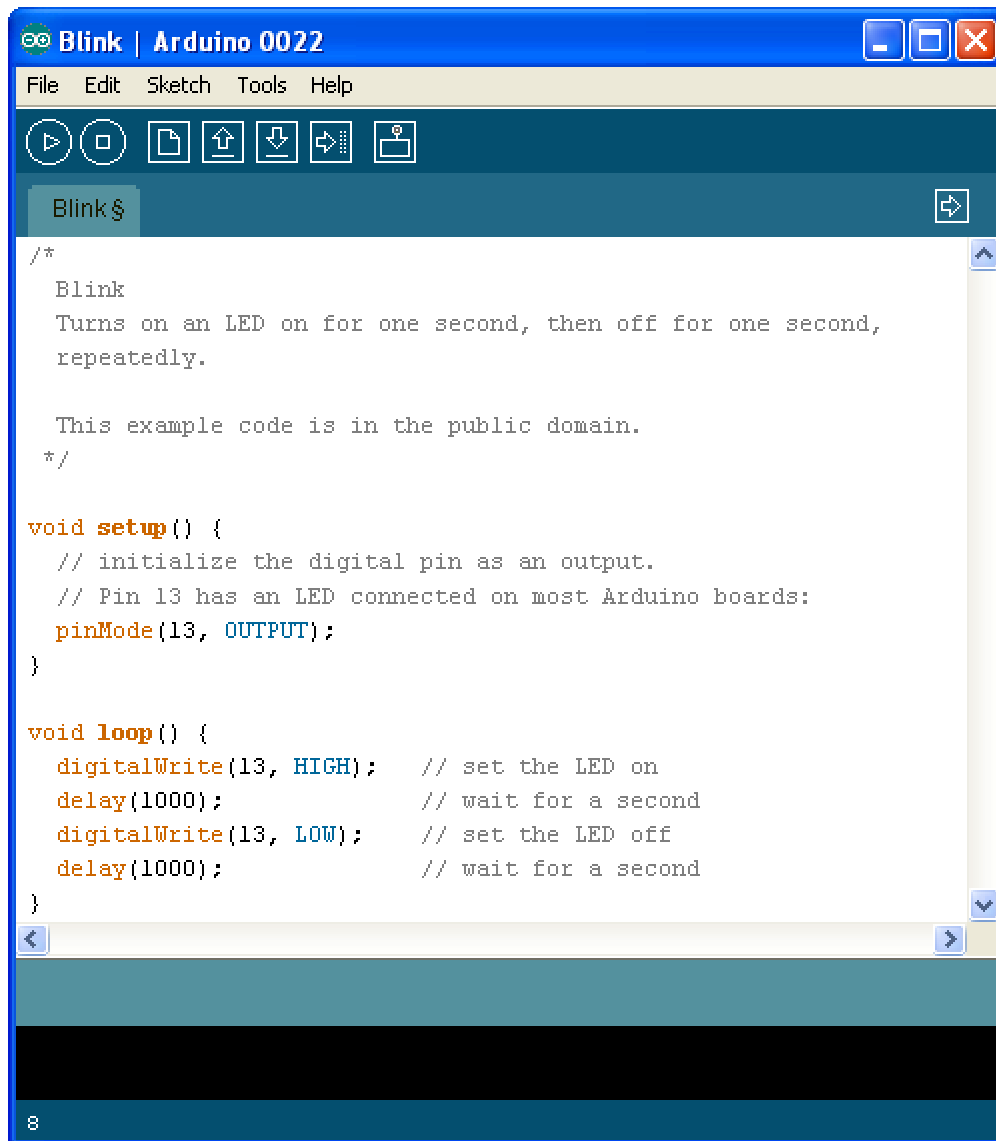
On teste le matériel en chargeant un programme d'exemple que l'on utilisera pour tester la carte.

On choisira un exemple qui consiste à faire clignoter une LED. Son nom est Blink et il se trouve dans la catégorie Basics :



Ouvrir le programme Blink

Une nouvelle fenêtre apparaît avec le programme Blink.



Avant d'envoyer le programme Blink vers la carte, il faut dire au logiciel quel est le nom de la carte et sur quel port elle est branchée. Pour cela, allez dans le menu "Tools" ("outils" en français) puis dans "Board" ("carte" en français). Vérifiez que c'est bien le nom "Arduin Uno" qui est coché. Si ce n'est pas le cas, cochez-le.

Blink | Arduino 0022

File Edit Sketch Tools Help

Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Serial Monitor Ctrl+Maj+M
Board
Serial Port
Burn Bootloader

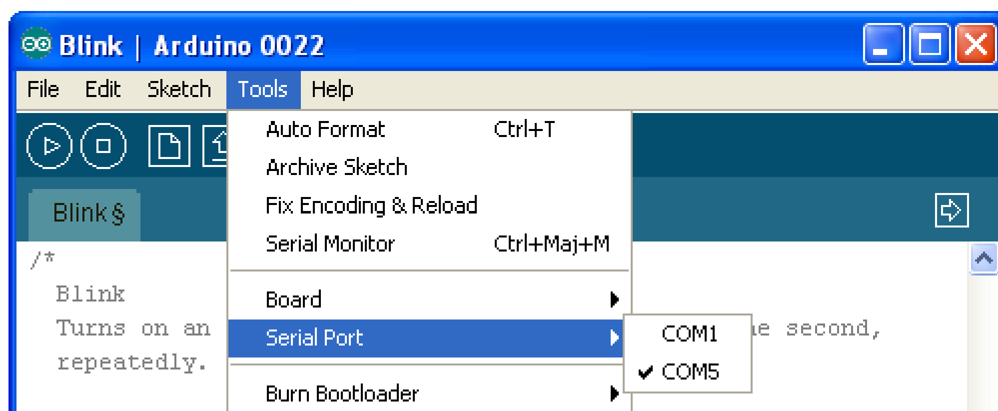
Arduino Uno
Arduino Duemilanove or Nano w/ ATmega328
Arduino Diecimila, Duemilanove, or Nano w/ ATmega168
Arduino Mega 2560
Arduino Mega (ATmega1280)
Arduino Mini
Arduino Fio
Arduino BT w/ ATmega328
Arduino BT w/ ATmega168
LilyPad Arduino w/ ATmega328
LilyPad Arduino w/ ATmega168
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega328
Arduino Pro or Pro Mini (5V, 16 MHz) w/ ATmega168
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega328
Arduino Pro or Pro Mini (3.3V, 8 MHz) w/ ATmega168
Arduino NG or older w/ ATmega168
Arduino NG or older w/ ATmega8

Blink §

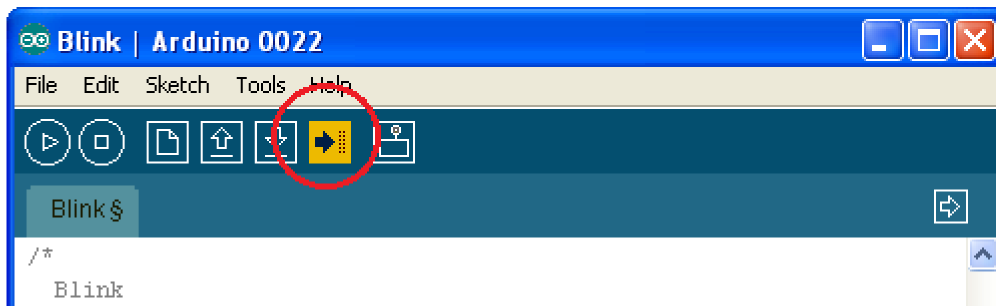
```
/*
  Blink
  Turns on an
  repeatedly.

  This example code is in the public domain
 */

void setup() {
  // initialize the digital pin as an outpu
  // Pin 13 has an LED connected on most A
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED
  delay(1000);              // wait for a s
  digitalWrite(13, LOW);    // set the LED
  delay(1000);              // wait for a s
}
```

8

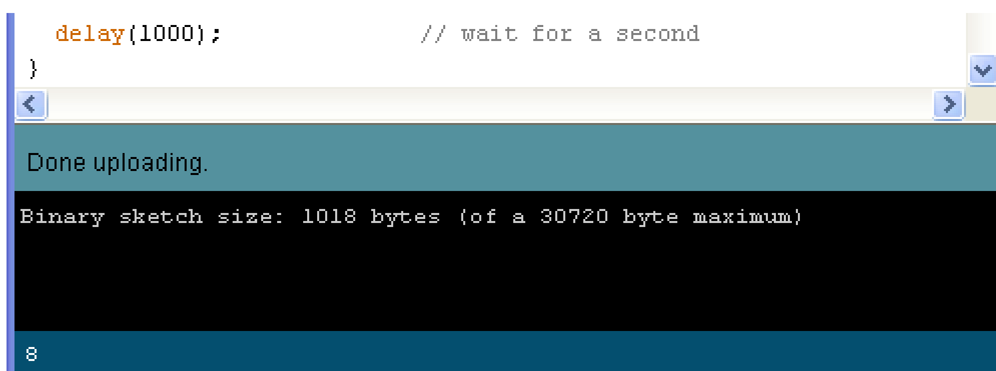
Allez ensuite dans le menu Tools, puis Serial port. Choisissez le port COMX, X étant le numéro du port qui est affiché. Ne choisissez pas COM1 car il n'est quasiment jamais connecté à la carte. Dans l'exemple, il s'agit de COM5 :



Maintenant, il va falloir envoyer le programme dans la carte. Pour ce faire, il suffit de cliquer sur le bouton Upload (ou "Télécharger" en Français), en jaune-orangé sur la photo :



En bas dans l'image, vous voyez le texte : "Uploading to I/O Board...", cela signifie que le logiciel est en train d'envoyer le programme dans la carte. Une fois qu'il a fini, il affiche un autre message :



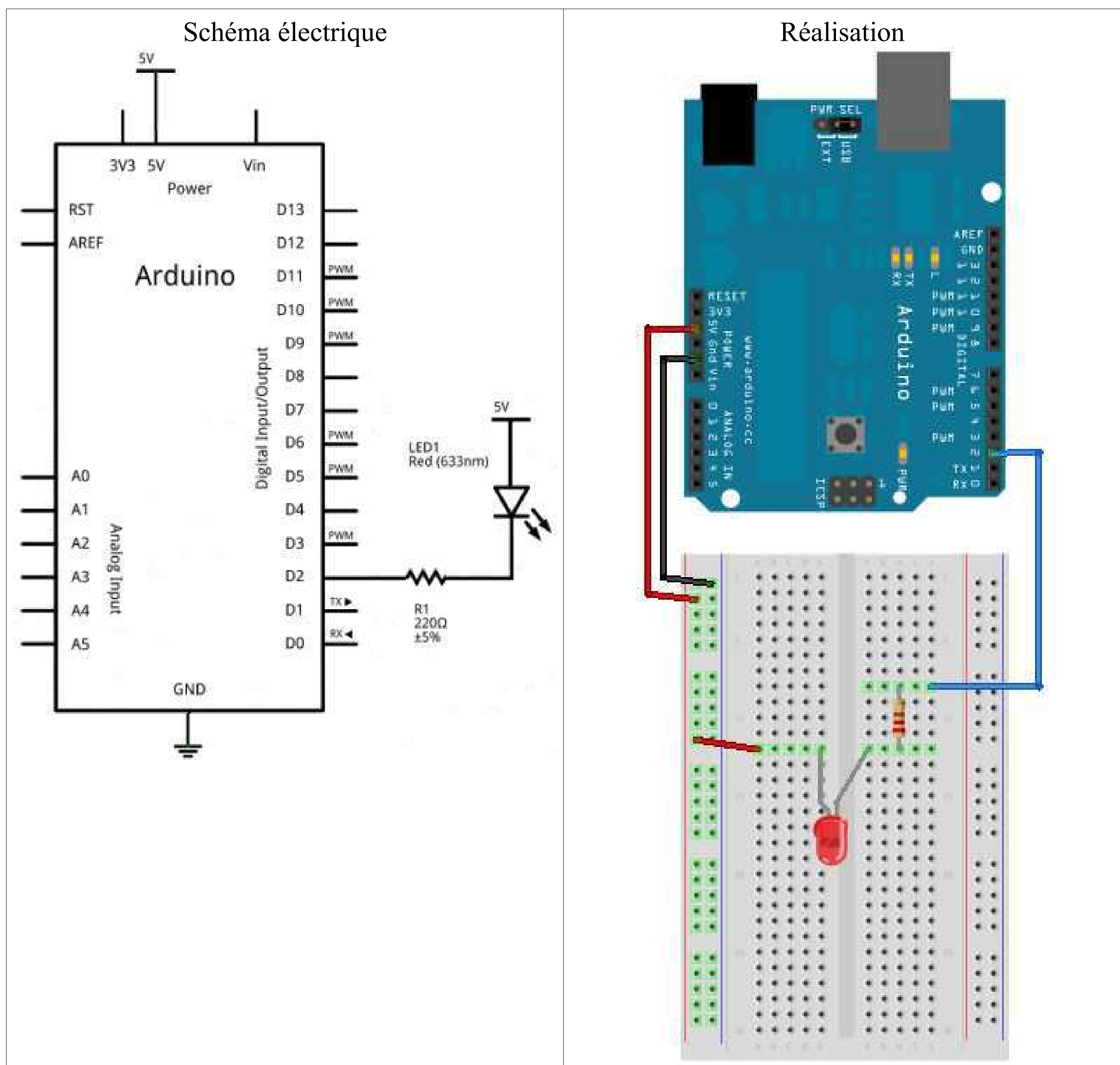
Le message afficher : "Done uploading" signale que le programme à bien été chargé dans la carte. Si votre matériel fonctionne, vous devriez avoir une LED sur la carte qui clignote :

4. Gestion des entrées / sorties

4.1. La diode électroluminescente

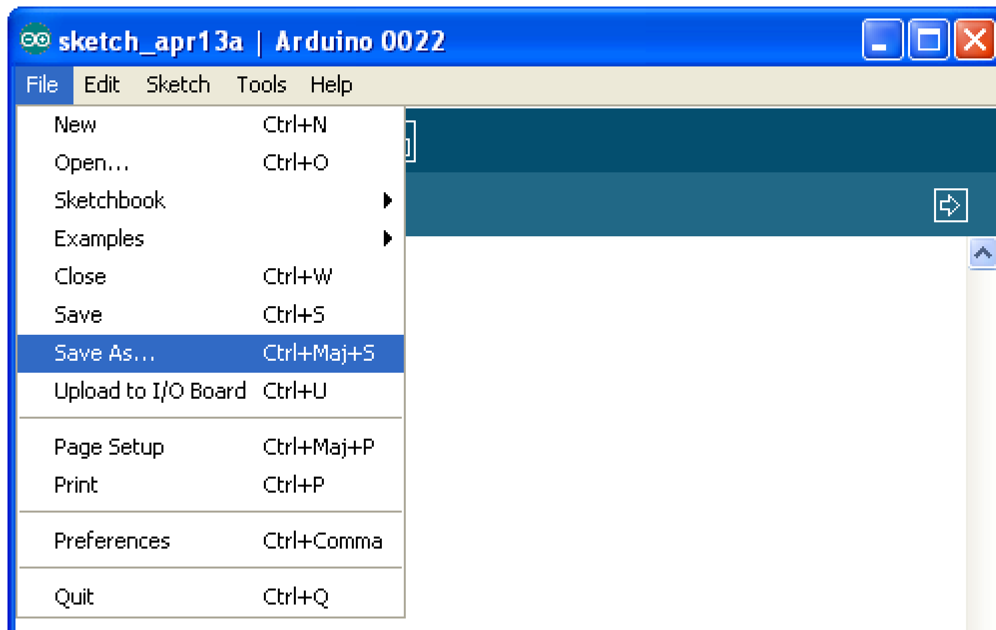
L'objectif de ce premier programme va consister à allumer une LED.

Avec le brochage de la carte Arduino, vous devrez connecter la plus grande patte au +5V (broche 5V). La plus petite patte étant reliée à la résistance, elle-même reliée à la broche numéro 2 de la carte. On pourrait faire le contraire, brancher la LED vers la masse et l'allumer en fournissant le 5V depuis la broche de signal. Cependant, les composants comme les microcontrôleurs n'aiment pas trop délivrer du courant, ils préfèrent l'absorber. Pour cela, on préférera donc alimenter la LED en la placant au +5V et en mettant la broche de Arduino à la masse pour faire passer le courant.

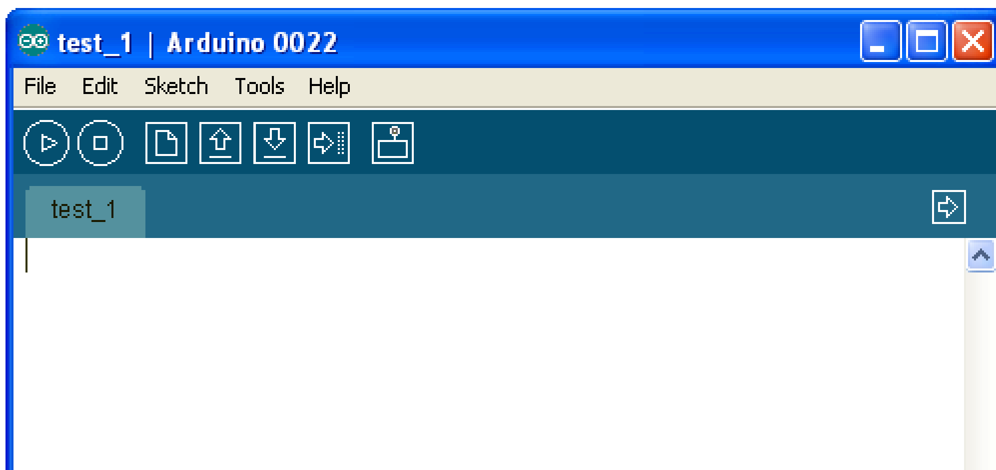


Pour programmer la carte, il faut créer un nouveau programme. Allez dans le menu File Et

choisissez l'option Save as... :



Tapez le nom du programme, puis Enregistrez. Vous arrivez dans votre nouveau programme, qui est vide pour l'instant, et dont le nom s'affiche en Haut de la fenêtre et dans un petit onglet :



Pour commencer le programme, il faut un code minimal. Ce code va permettre d'initialiser la carte :

```
void setup()    //fonction d'initialisation de la carte
{
        //contenu de l'initialisation
}
void loop()     //fonction principale, elle se répète (s’exécute) à l'infini
{
        //contenu du programme
}
```

Il faut avant tout définir les broches du micro-contrôleur. Cette étape constitue elle-même deux sous étapes. La première étant de créer une variable définissant la broche utilisée, ensuite, définir si la broche utilisée doit être une entrée du micro-contrôleur ou une sortie.

Premièrement, définissons la broche utilisée du micro-contrôleur :

```
const int led_rouge = 2;   //définition de la broche 2 de la carte en tant que variable
```

Il faut maintenant dire si cette broche est une entrée ou une sortie. Cette ligne de code doit se trouver dans la fonction setup(). La fonction à utiliser est pinMode(). Pour utiliser cette fonction, il faut lui envoyer deux paramètres :

- Le nom de la variable que l'on a défini à la broche
- Le type de broche que cela va être (entrée ou sortie)

```
void setup()
{
        pinMode(led_rouge, OUTPUT);      // initialisation de la broche 2 comme étant une sortie
}
```

La deuxième étape consiste à créer le contenu du programme. Celui qui va aller remplacer le commentaire dans la fonction loop() pour allumer la LED.

On va utiliser la fonction digitalWrite() qui va écrire une valeur HAUTE (+5V) ou BASSE (0V) sur une sortie numérique. La LED étant connecté au pôle positif de l'alimentation, il faut qu'elle soit reliée au 0V. Par conséquent, on doit mettre un état bas sur la broche du microcontrôleur. Ainsi, la différence de potentiel aux bornes de la LED permettra à celle-ci de s'allumer

La fonction digitalWrite() requiert deux paramètres : le nom de la broche que l'on veut mettre à un état logique et la valeur de cet état logique.

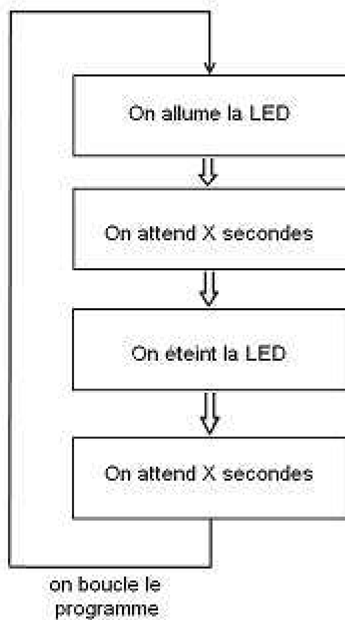
Voici le code entier :

```
const int led_rouge = 2;        //définition de la broche 2 de la carte en tant que variable

void setup()    //fonction d'initialisation de la carte
{
        pinMode(led_rouge, OUTPUT);      //initialisation de la broche 2 comme étant une sortie
}

void loop()      //fonction principale, elle se répète (s'exécute) à l'infini
{
        digitalWrite(led_rouge, LOW);        // écriture en sortie (broche 2) d'un état BAS
}
```

4.2. Temporisation

4.2.1. Fonction delay()

La fonction delay() va servir à mettre en pause le programme pendant un temps prédéterminé. Elle admet un paramètre qui est le temps pendant lequel on veut mettre en pause le programme. Ce temps doit être donné en millisecondes.



Pour faire clignoter la LED on fait intervenir la fonction delay(), qui va mettre le programme en pause pendant un certain temps.

Ensuite, on éteint la LED.

On met en pause le programme.

Puis on revient au début du programme. On recommence et ainsi de suite.

Ce qui donne le code suivant :

```
const int led_rouge = 2;        //définition de la broche 2 de la carte en tant que variable

void setup()     //fonction d'initialisation de la carte
{
        pinMode(led_rouge, OUTPUT);      //initialisation de la broche 2 comme étant une sortie
}

void loop()      //fonction principale, elle se répète (s'exécute) à l'infini
{
        digitalWrite(led_rouge, LOW);       // allume la LED
        delay(1000);                        // fait une pause de 1 seconde

        digitalWrite(led_rouge, HIGH);      // éteint la LED
        delay(1000);                        // fait une pause de 1 seconde
}
```

Voir le résultat final :

4.2.2. Fonction millis()

À l'intérieur du cœur de la carte Arduino se trouve un chronomètre. Ce chrono mesure l'écoulement du temps depuis le lancement de l'application dont la granularité est la milliseconde. La fonction millis() retourne la valeur courante de ce compteur qui est capable de mesurer une durée allant jusqu'à 50 jours.

Si on veut faire clignoter une LED et faire avancer un robot par exemple, on ne peut pas utiliser la fonction delay() qui met en pause le programme.

Avec la fonction milli(), le programme n'est plus bloquant. En mettant en place un bouton de détection de collisions, l'état du bouton sera vérifié très fréquemment ce qui permet de s'assurer que si jamais on rentre dans un mur, on coupe très vite les moteurs du robot.

Voici le code :

```
const int     moteur = 3;          // moteur connecté à la broche 3
const int     led = 2;             // led connectée à la broche 2
const int     bouton = 7;          // bouton connecté à la broche 7

long          temps = millis();    // variable qui stocke la mesure du temps
boolean       etat_led = 0;        // par défaut la LED sera éteinte

void setup()
{
        // définition des entrées/sorties
        pinMode(moteur, OUTPUT);
        pinMode(led, OUTPUT);
        pinMode(bouton, INPUT);

        digitalWrite(moteur, HIGH);         //on met le moteur en marche

        digitalWrite(led, etat_led);        //on éteint la LED
}

void loop()
{
        if ( digitalRead(bouton) == HIGH )  //si le bouton est cliqué (on rentre dans un mur)
        {
                digitalWrite(moteur, LOW);  //on arrête le moteur
        }
        else    //sinon on clignote
        {
                //on compare l'ancienne valeur du temps et la valeur sauvée
                //si la comparaison (l'un moins l'autre) dépasse 1000...
                //...cela signifie qu'au moins une seconde s'est écoulée
                if ( (millis() - temps) > 1000 )
                {
                        etat_led = !etat_led;           //on inverse l'état de la LED
                        digitalWrite(led, etat_led);    //on allume ou éteint

                        temps = millis();               //on stocke la nouvelle heure
                }
        }
}
```

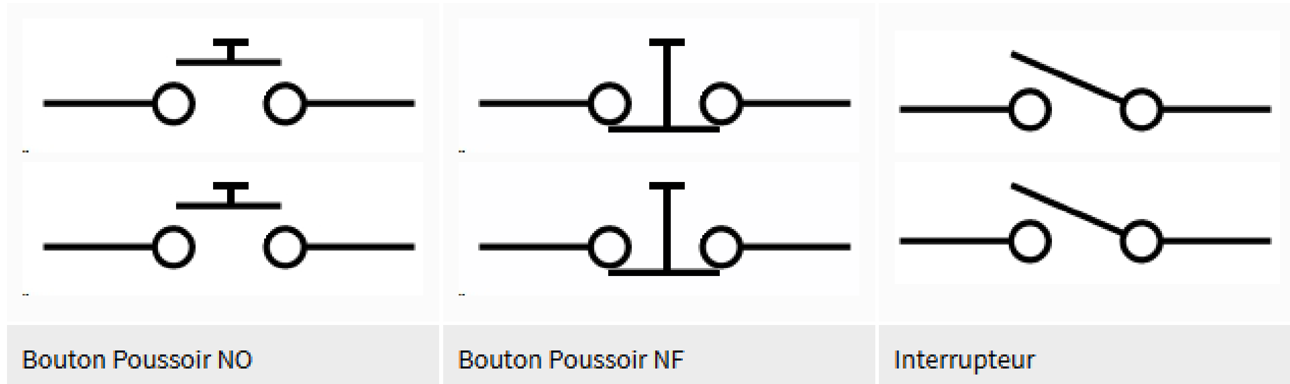
Remarque : il n'est pas possible de tester la condition "millis() - temp == 1000" car cela signifierait que l'on vérifie que 1000 millisecondes EXACTEMENT se sont écoulées, ce qui est très peu probable à cause du temps d'exécution du programme.

4.3. Le bouton poussoir

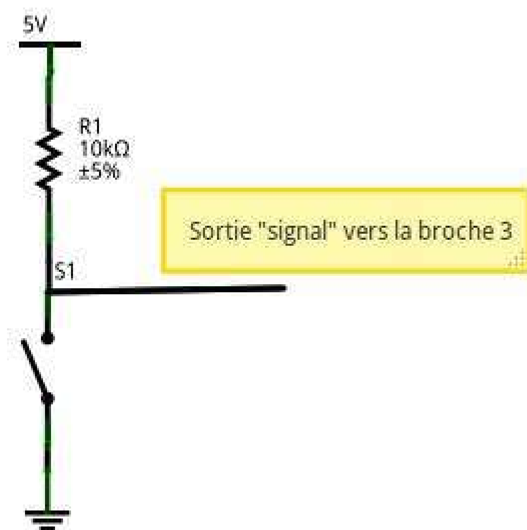
Les boutons poussoirs (BP) normalement ouvert (NO) ont deux positions :

- Relâché : le courant ne passe pas, le circuit est "ouvert".
- Appuyé : le courant passe, le circuit est fermé.

Le bouton poussoir normalement fermé (NF) est l'opposé du type précédent, c'est-à-dire que lorsque le bouton est relâché, il laisse passer le courant. Et inversement :



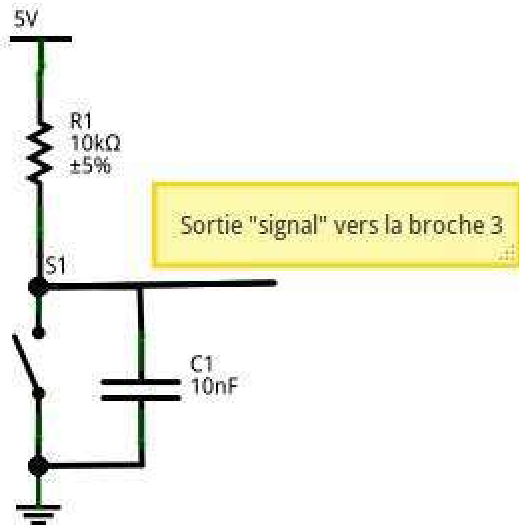
À la différence d'un bouton poussoir, l'interrupteur agit comme une bascule. Un appui ferme le circuit et il faut un second appui pour l'ouvrir de nouveau. Il possède donc deux états stables (ouvert ou fermé). On dit qu'un interrupteur est bistable.



En électronique, on a toujours des perturbations (générées par des lampes à proximité, un téléphone portable, ...). On appelle ça des contraintes de CEM. Pour contrer ces effets nuisibles, ont place en série avec le bouton une résistance de pull-up. Cette résistance sert à "tirer" ("to pull" in english) le potentiel vers le haut (up) afin d'avoir un signal clair sur la broche étudiée.

Sur le schéma suivant, on voit ainsi qu'en temps normal le "signal" à un potentiel de 5V. Ensuite, lorsque l'utilisateur appuiera sur le bouton une connexion sera faite avec la masse. On lira alors une valeur de 0V pour le signal. Voici donc un deuxième intérêt de la résistance de pull-up, éviter le court-circuit qui serait généré à l'appui !

Les boutons ne sont pas des systèmes mécaniques parfaits. Du coup, lorsqu'un appui est fait dessus, le signal ne passe pas immédiatement et proprement de 5V à 0V. En l'espace de quelques millisecondes, le signal va "sauter" entre 5V et 0V plusieurs fois avant de se stabiliser. Il se passe le même phénomène lorsque l'utilisateur relâche le bouton. Ce genre d'effet n'est pas désirable, car il peut engendrer des parasites au sein du programme.



Pour atténuer ce phénomène, on utilise un condensateur en parallèle avec le bouton. Ce composant servira ici "d'amortisseur" qui absorbera les rebonds (comme sur une voiture avec les cahots de la route). Le condensateur, initialement chargé, va se décharger lors de l'appui sur le bouton. S'il y a des rebonds, ils seront encaissés par le condensateur durant cette décharge. Il se passera le phénomène inverse (charge du condensateur) lors du relâchement du bouton.

Des résistances de pull-up existent aussi en interne du microcontrolleur de l'Arduino, ce qui évite d'avoir à les rajouter. Ces dernières ont une valeur de 20 kΩ. Elles peuvent être utilisées sans aucune contraintes techniques. Cependant, si vous les mettez en marche, il faut se souvenir que cela équivaut à mettre la broche à l'état haut (et en entrée évidemment). Donc si vous repassez à un état de sortie ensuite, rappelez vous bien que tant que vous ne l'avez pas changée elle sera à l'état haut.

Pour récupérer l'appui du bouton, on doit lire l'état d'une entrée numérique.

Schéma électrique	Réalisation

Lorsque le bouton est relâché, la tension à ses bornes sera de +5V, donc un état logique HIGH. S'il est appuyé, elle sera de 0V, donc LOW.

La fonction digitalRead() permet de lire l'état logique d'une entrée logique. Cette fonction prend un paramètre qui est la broche à tester et elle retourne une variable de type int.

Le programme suivant allume une LED lorsque le bouton est appuyé. Lorsque l'on relâche le bouton, la LED doit s'éteindre.

```
const int       bouton = 2;     //le bouton est connecté à la broche 2 de la carte Adruino
const int       led = 13;       //la LED à la broche 13

int             etatBouton;     //variable qui enregistre l'état du bouton

void setup()
{
        pinMode(led, OUTPUT);           //la led est une sortie
        pinMode(bouton, INPUT);         //le bouton est une entrée
        etatBouton = HIGH;              //on initialise l'état du bouton comme "relaché"
}

void loop()
{
        etatBouton = digitalRead(bouton);       //Rappel : bouton = 2

        if ( etatBouton == HIGH )       //test si le bouton a un niveau logique HAUT
        {
                digitalWrite(led, HIGH);        //la LED reste éteinte
        }
        else    //test si le bouton a un niveau logique différent de HAUT (donc BAS)
        {
                digitalWrite(led, LOW);         //le bouton est appuyé, la LED est allumée
        }
}
```

Voir le résultat final :

4.4. Les interruptions matérielles

Comme vous l'avez remarqué dans la partie précédente, pour récupérer l'état du bouton il faut surveiller régulièrement l'état de ce dernier. Cependant, si le programme a quelque chose de long à traiter, l'appui sur le bouton ne sera pas très réactif et lent à la détente.

Pour pallier ce genre de problème, les constructeurs de microcontrôleurs ont mis en place des systèmes qui permettent de détecter des évènements et d'exécuter des fonctions dès la détection de ces derniers.

Une interruption est un déclenchement qui arrête l'exécution du programme pour faire une tâche demandée. Dans le cas d'une carte Arduino UNO, on trouve deux broches pour gérer des interruptions externes (qui ne sont pas dues au programme lui même), la 2 et la 3. Pour déclencher une interruption, plusieurs cas de figure sont possibles :

- LOW : Passage à l'état bas de la broche
- FALLING : Détection d'un front descendant (passage de l'état haut à l'état bas)

- RISING : Détection d'un front montant (pareil qu'avant, mais dans l'autre sens)
- CHANGE : Changement d'état de la broche

Autrement dit, s'il y a un changement d'un type énuméré au-dessus, alors le programme sera interrompu pour effectuer une action. Lorsque l'interruption aura été exécutée et traitée, il reprendra comme si rien ne s'était produit.

La fonction attachInterrupt(interrupt, function, mode) accepte trois paramètres :

1. interrupt : numéro de la broche utilisée pour l'interruption (0 pour la broche 2 et 1 pour la broche 3)
2. function : nom de la fonction à appeler lorsque l'interruption est déclenchée
3. mode : type de déclenchement (cf. ci-dessus)

Pour appeler une fonction nommée Reagir() lorsque l'utilisateur appuie sur un bouton branché sur la broche 2 le code sera :

attachInterrupt(0, Reagir, FALLING);

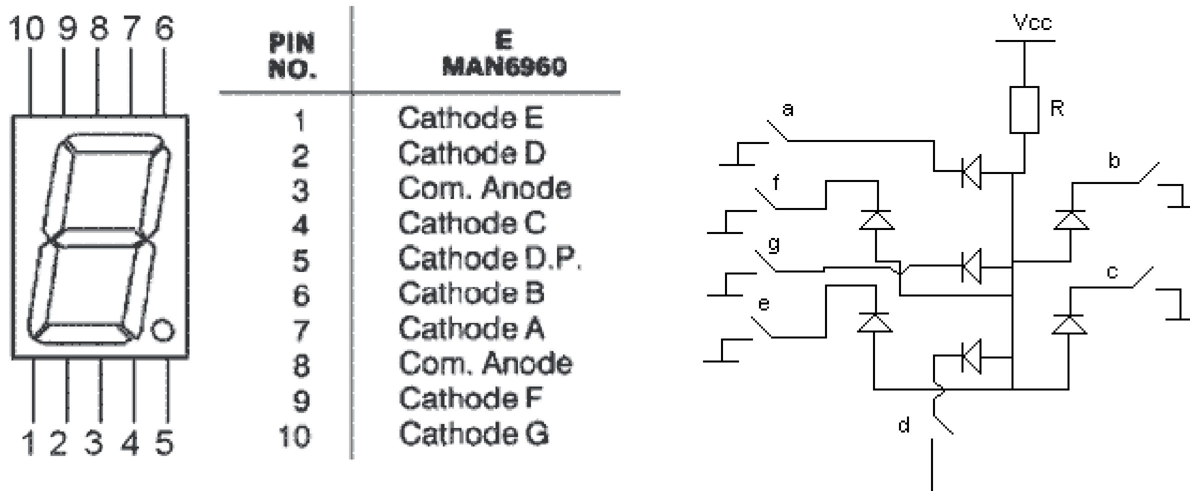
NB : la fonction Reagir() ne peut pas prendre d'argument et ne retournera aucun résultat.

Attention :

- Les interruptions mettent le programme en pause, et une mauvaise programmation peut entraîner une altération de l'état de vos variables.
- Les fonctions delay() et millis() n'auront pas un comportement correct. En effet, pendant ce temps le programme principal est complètement stoppé, donc les fonctions gérant le temps ne fonctionneront plus, elles seront aussi en pause et laisseront la priorité à la fonction d'interruption. La fonction delay() est donc désactivée et la valeur retournée par millis() ne changera pas.

4.5. Afficheurs 7 segments

Un afficheur 7 segments possède... 7 LED encastrées dans un boiter. On dénombre donc 8 portions en comptant le point décimal (DP) de l'afficheur.



PIN NO.	E MAN6960
1	Cathode E
2	Cathode D
3	Com. Anode
4	Cathode C
5	Cathode D.P.
6	Cathode B
7	Cathode A
8	Com. Anode
9	Cathode F
10	Cathode G

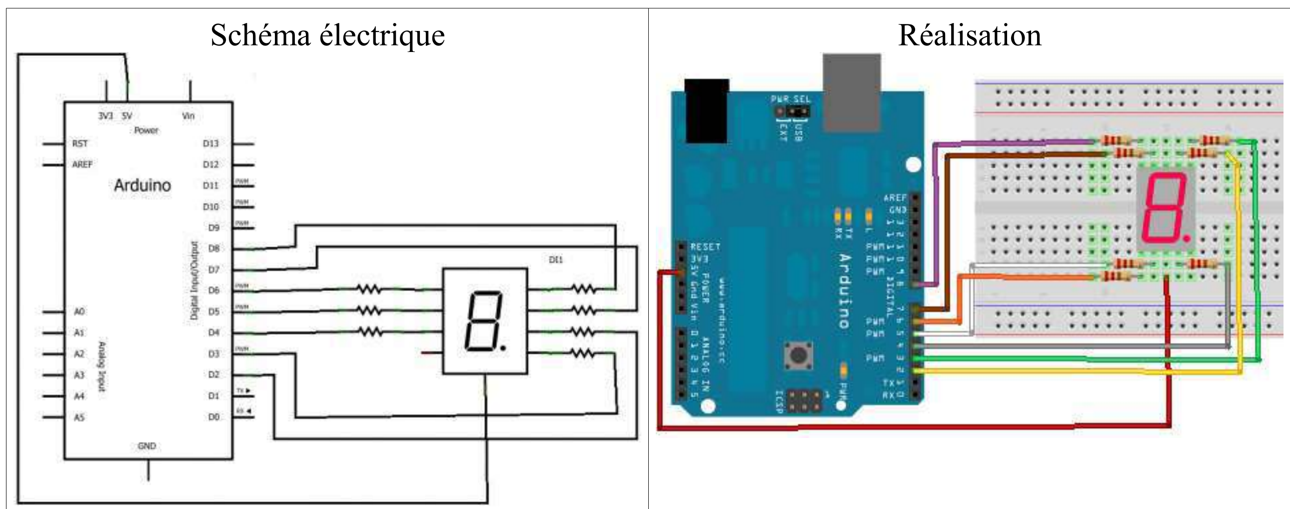
Les interrupteurs a,b,c,d,e,f,g représentent les signaux pilotant chaque segments.

Il faut ajouter une résistance de limitation de courant entre la broche isolée et la broche de signal. Traditionnellement, on prendra une résistance de 330 Ω pour une tension de +5V. Si vous voulez augmenter la luminosité, il suffit de diminuer cette valeur. Si au contraire vous voulez diminuer la luminosité, augmenter la résistance.

Voici un tableau illustrant les caractères possibles et quels segments allumés :

Caractère	seg. A	seg. B	seg. C	seg. D	seg. E	seg. F	seg. G
0	x	x	x	x	x	x	
1		x	x				
2	x	x		x	x		x
3	x	x	x	x			x
...							

Pour allumer les LED, il suffit de positionner la sortie numérique de la carte Arduino sur la bonne valeur.



Voici le code pour afficher '5' :

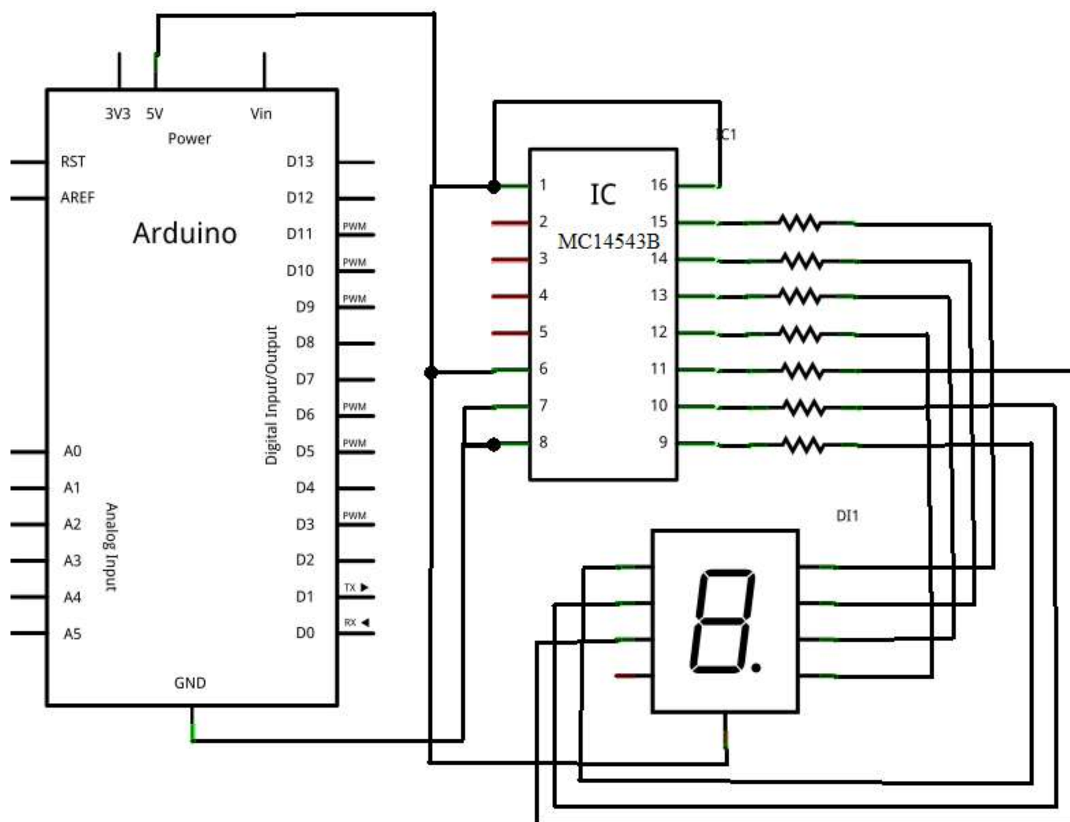
```
int     cinq[] = {LOW, HIGH, LOW, LOW, HIGH, LOW, LOW};

void setup()
{
        //notez que l'on ne gère pas l'affichage du point
        int     i;
        for (i = 2 ; i <= 8 ; i++)
        {
                //définition des broches en sortie (2 à 8)
                pinMode(i, OUTPUT);

                //mise à l'état HAUT de ces sorties pour éteindre les LED de l'afficheur
                digitalWrite(i, HIGH);
        }
}

void loop()
```

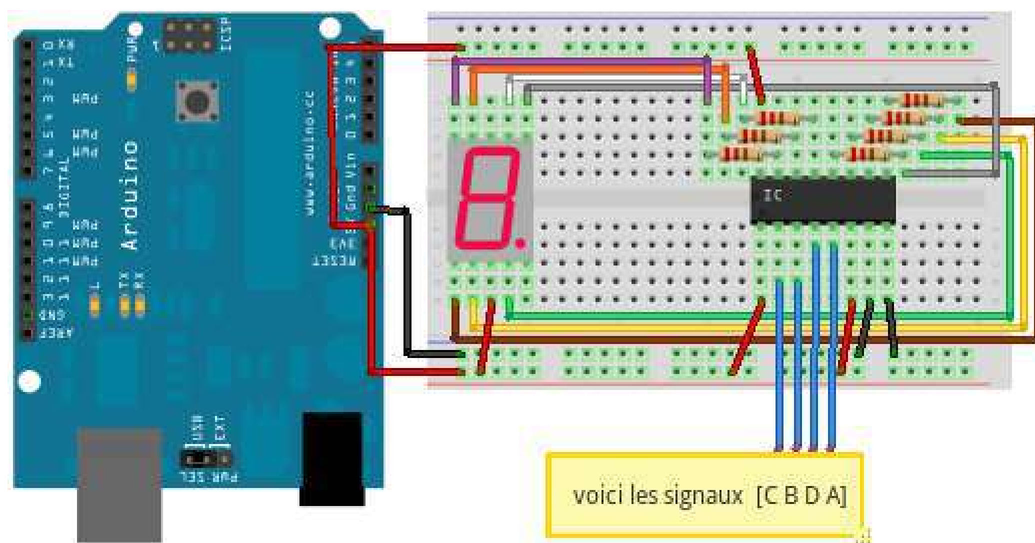
```
{
	//affichage du chiffre 5, d'après le tableau précédent
	int	i, j = 0;
	for (i = 2 ; i <= 8 ; i++, j++)
	{
		digitalWrite(i, cinq[j]);
	}
}
```

Cependant, pour représenter les chiffres de 0 à 9 il suffit de 4 bits et non des 7 sorties numériques de la carte. Pour cela, on utilise un décodeur BCD (ex : le MC14543B).

Ce composant est monté sur un DIP 16 (DIP : Dual Inline Package) et possède les fonctionnaalités suivantes :

- Si l'on met la broche Bl (Blank, n°7) à un, toutes les sorties passent à zéro. Pour ne pas l'utiliser il faut la connecter à la masse pour la désactiver.
- Les entrées A, B, C et D (broches 5,3,2 et 4 respectivement) sont actives à l'état HAUT. Les sorties elles sont actives à l'état BAS (pour piloter un afficheur à anode commune) OU HAUT selon l'état de la broche PH (6).
- La broche BI/RBO (n°4) sers à inhiber les entrées. On ne s'en servira pas et donc on la mettra à l'état HAUT (+5V)
- LD (n°1) sert à faire une mémoire de l'état des sorties.
- Les deux broches d'alimentation sont la 8 (GND/VSS, masse) et la 16 (VCC, +5V)



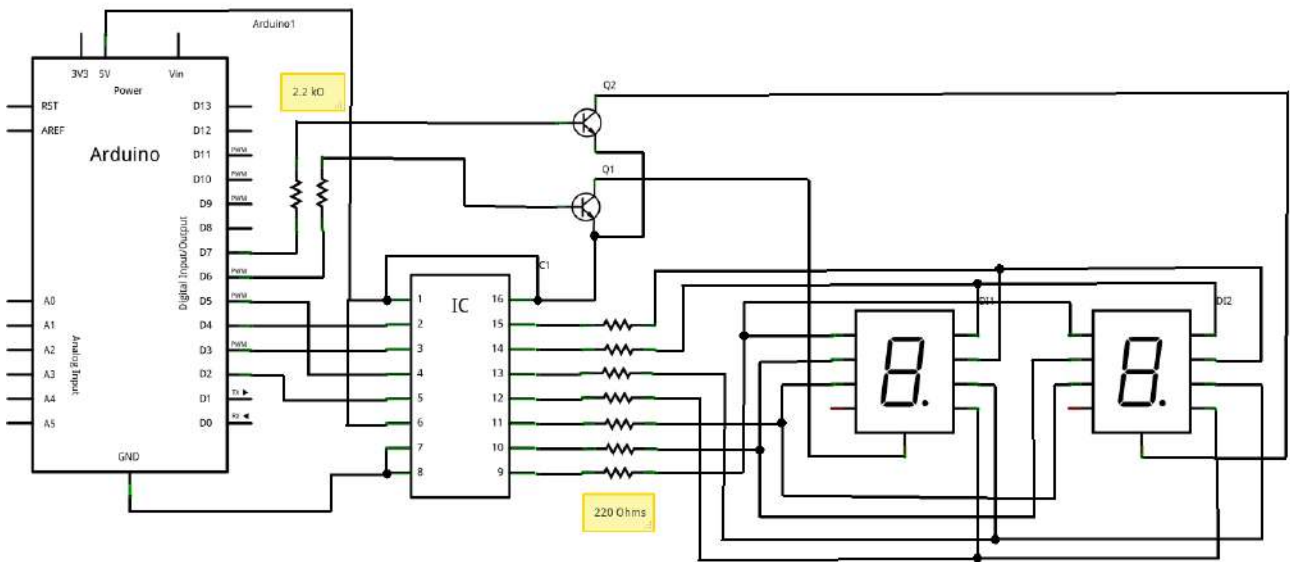


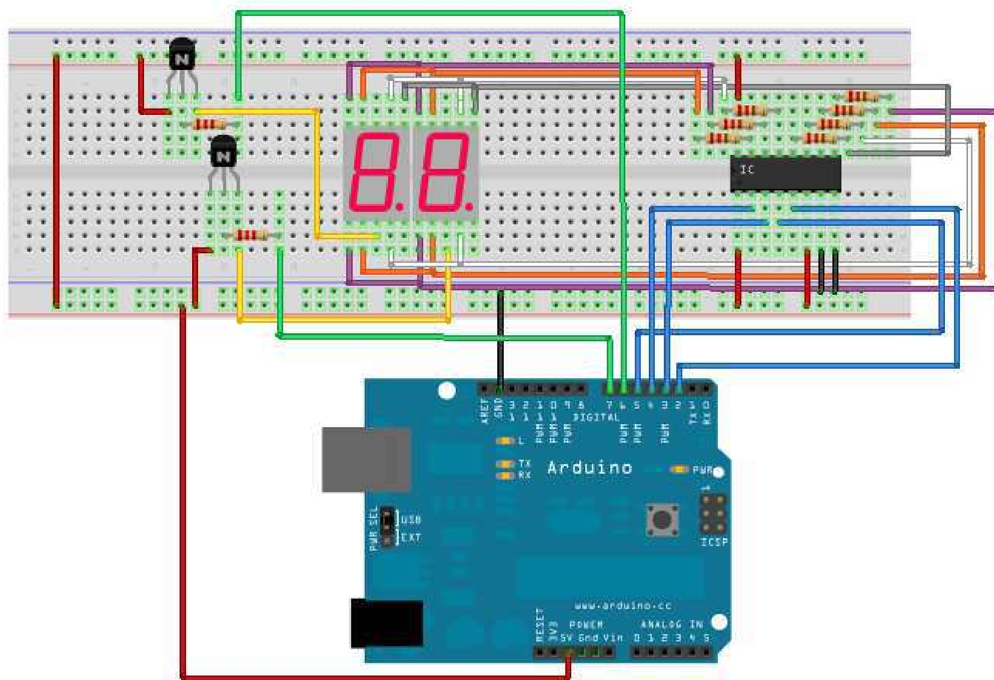
Voici le code pour faire afficher les chiffres de 0 à 9 :

```
void setup()
{
        int     i;
        for (i = 2 ; i <= 5 ; i++)
        {
                //définition des broches en sortie (2 à 5)
                pinMode(i, OUTPUT);
        }
}
void loop()
{
        char    i;              //variable "compteur"
        for (i=0; i < 10; i++)
        {
                afficher(i);    //on appel la fonction d'affichage
                delay(1000);    //on attend 1 seconde
        }
}
void afficher(char chiffre)
{
         //mise à zéro tout les segments
        int     i;
        for (i = 2 ; i <= 5 ; i++)
        {
                digitalWrite(i, LOW);
        }

        //On compte en binaire à l'aide d'un masque
        char    mask = 1;
        for (i = 2 ; i <= 5 ; i++)
```

```
	{
		int nl = ( chiffre & mask ) ? HIGH : LOW;
		digitalWrite(i, nl);
		mask *= 2 ;
	}
}
```

Pour utiliser deux afficheurs, on ne dispose que de 6 broches. Pour réduire le nombre de broches, on utilise un décodeur BCD, ce qui ferait 4 broches par afficheurs, soit 8 broches au total. Pour afficher des chiffres différents sur chaque afficheur il faut allumer un afficheur et d'éteindre l'autre tout en les connectant ensemble sur le même décodeur. Cette opération est en fait un clignotement de chaque afficheur par alternance à l'aide de 2 transistors bipolaires (type NPN 2N2222) pilotés par la carte.





Exemple : compteur de 0 à 99 sur 2 segments.

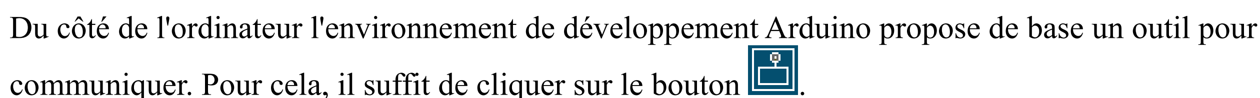
```
//définitions des broches des transistors pour chaque afficheur (dizaines et unités)
const int       alim_dizaine = 6;
const int       alim_unite = 7;

bool            afficheur = false;              //variable pour le choix de l'afficheur

void setup()
{
        int     i;
        for (i = 2 ; i <= 5 ; i++)
        {
                pinMode(i, OUTPUT);             //définition des broches en sortie (2 à 5)
                digitalWrite(i, LOW);           //Les broches sont toutes mises à l'état bas
        }

        pinMode(alim_dizaine, OUTPUT);
        pinMode(alim_unite, OUTPUT);
        digitalWrite(alim_dizaine, LOW);
        digitalWrite(alim_unite, LOW);
}

void loop()
{
        //gestion du rafraichissement
        //si plus de 10 ms qu'on affiche, on change de 7 segments (alternance unité <-> dizaine)
        if ( (millis() - temps) > 10 )
        {
                //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou dizaine)
                afficheur = !afficheur;

                //on affiche la valeur sur l'afficheur
                //afficheur : true->dizaines, false->unités
                afficher_nombre(valeur, afficheur);

        temps = millis();       //on met à jour le temps
        }

        //ici, on peut traiter les évènements (bouton...)
}

//fonction permettant d'afficher un nombre
//elle affiche soit les dizaines soit les unités
void afficher_nombre(char nombre, bool afficheur)
{
        char unite = 0, dizaine = 0;

        if ( nombre > 9 )
                dizaine = nombre / 10;          //on recupere les dizaines
        unite = nombre – (dizaine*10);          //on recupere les unités
```

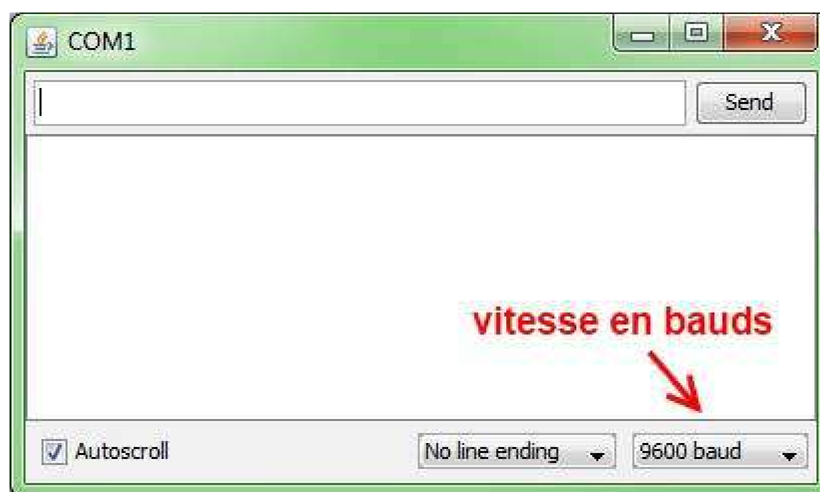
```
        //affichage dizaines
        if ( afficheur )
        {
                //on affiche les dizaines
                digitalWrite(alim_unite, LOW);
                afficher(dizaine);
                digitalWrite(alim_dizaine, HIGH);
        }
        else
        {
                //on affiche les unités
                digitalWrite(alim_dizaine, LOW);
                afficher(unite);
                digitalWrite(alim_unite, HIGH);
        }
}
//fonction écrivant sur un seul afficheur
void afficher(char chiffre)
{
         //mise à zéro tout les segments
        int     i;
        for (i = 2 ; i <= 5 ; i++)
        {
                digitalWrite(i, LOW);
        }

        //On compte en binaire à l'aide d'un masque
        char    mask = 1;
        for (i = 2 ; i <= 5 ; i++)
        {
                int nl = ( chiffre & mask ) ? HIGH : LOW;
                digitalWrite(i, nl);
                mask *= 2 ;
        }
}
```

Voir le résultat final :

Remarque : Si vous voulez tester le phénomène de persistance rétinienne, vous pouvez changer le temps de la boucle de rafraichissement. Si vous l'augmenter, vous commencerez à voir les afficheurs clignoter.

5. Communication par la liaison série

Du côté de l'ordinateur l'environnement de développement Arduino propose de base un outil pour communiquer. Pour cela, il suffit de cliquer sur le bouton .



le terminal série

Dans cette fenêtre, vous allez pouvoir envoyer et recevoir des messages sur la liaison série de votre ordinateur (qui est émulée par l'Arduino).

Du côté du programme, pour utiliser la liaison série et communiquer avec l'ordinateur, on utilise un objet qui est intégré nativement dans l'ensemble Arduino : l'objet Serial.

Cet objet rassemble des informations (vitesse, bits de données, etc.) et des fonctions (envoi, lecture de réception,...) sur ce qu'est une voie série pour Arduino.

Pour commencer, il faut initialiser l'objet Serial afin de définir la vitesse de communication entre l'ordinateur et la carte Arduino grâce à la fonction begin(). Cette vitesse doit être identique côté ordinateur et côté programme.

Exemple : Serial.begin(9600);   //établissement d'une communication série à 9600 bauds

5.1. Envoi de données

La fonction print() permet d'envoyer des caractères.

Le programme ci-dessous envoi l'ensemble des lettres de l'alphabet.

```
void setup()
{
	Serial.begin(9600);	//établissement d'une communication série à 9600 bauds
}
void loop()
{
	char i = 0;
	char lettre = 'a';	// ou 'A' pour envoyer en majuscule
```

```
        Serial.println("------  L'alphabet ------");       //petit message d'accueil
        //on commence les envois
        for (i=0; i<26; i++)
        {
                Serial.print(lettre);    //on envoie la lettre
                lettre = lettre + 1;     //on passe à la lettre suivante
                delay(250);              //on attend 250ms avant de réenvoyer
        }

        Serial.println("");      //on fait un retour à la ligne
        delay(2000);             //on attend 2 secondes avant de renvoyer l'alphabet
}
```

5.2. Reception des données

Pour vérifier si on a reçu des données, on va régulièrement interroger la carte pour lui demander si des données sont disponibles dans son buffer de réception. Un buffer est une zone mémoire permettant de stocker des données sur un cours instant. Cette mémoire est dédiée à la réception sur la voie série. Il en existe un aussi pour l'envoi de donnée, qui met à la queue leu leu les données à envoyer et les envoie dès que possible. En résumé, un buffer est une sorte de salle d'attente pour les données.

Pour vérifier si des données sont arrivées, on utilise la fonction available() (de l'anglais "disponible") de l'objet Serial. Cette fonction renvoie le nombre de caractères dans le buffer de réception de la liaison série ou -1 quand il n'y a rien à lire sur le buffer de réception.

Une fois que l'on sait qu'il y a des données, il faut aller les lire. La lecture se fera tout simplement avec la fonction... read() !

Cette fonction renverra le premier caractère arrivé non traité. On accède donc caractère par caractère aux données reçues. Si jamais rien n'est à lire, la fonction renverra -1 pour le signaler.

Exemple. L'utilisateur saisit un caractère à partir de l'ordinateur et si ce caractère est minuscule, il est renvoyé en majuscule ; s'il est majuscule il est renvoyé en minuscule. Enfin, si le caractère n'est pas une lettre on se contente de le renvoyer normalement, tel qu'il est.

```
int carlu;        //stock le caractère lu sur la voie série

void setup()
{
        Serial.begin(9600);     //démarre une nouvelle liaison série à 9600bauds
}

void loop()
{
        //on commence par vérifier si un caractère est disponible dans le buffer
        if ( Serial.available() > 0 )
        {
                carlu = Serial.read();  //lecture du premier caractère disponible

                //Est-ce que c'est un caractère minuscule ?
                if ( carlu >= 'a' && carlu <= 'z' )
```

```
		{
			carlu = carlu – 'a';	//on garde juste le "numéro de lettre"
			carlu = carlu + 'A';	//on passe en majuscule
		}
		else
			//Est-ce que c'est un caractère MAJUSCULE ?
			if (carlu >= 'A' && carlu <= 'Z' )
			{
				carlu = carlu – 'A';	//on garde juste le "numéro de lettre"
				carlu = carlu + 'a';	//on passe en minuscule
			}

		//ni l'un ni l'autre on renvoie en tant que BYTE
		//ou alors on renvoie le caractère modifié
		Serial.write(carlu);	// envoie le caractère en tant que variable de type byte
	}
}
```

Voir le résultat final :

Si vous voulez éviter de mettre le test de présence de données sur la voie série dans votre code, Arduino a rajouter une fonction qui s'exécute de manière régulière. Cette dernière se lance régulièrement avant chaque redémarrage de la loop. Ainsi, si vous n'avez pas besoin de traiter les données de la voie série à un moment précis, il vous suffit de rajouter cette fonction.

Pour l'implémenter c'est très simple, il suffit de mettre du code dans une fonction nommée serialEvent() qui sera à rajouter en dehors du setup et du loop. Le reste du traitement de texte se fait normalement, avec Serial.read par exemple.

Voici un exemple de squelette possible :

```
const int	Led = 11;	//on met une LED sur la broche 11

void setup()
{
	pinMode(Led, OUTPUT);		//la LED est une sortie
	digitalWrite(maLed, HIGH);	//on éteint la LED

	Serial.begin(9600);		//on démarre la voie série
}

void loop()
{
	delay(1000);			//fait une petite pause de 1 s
	//...on ne fait rien dans la loop
	digitalWrite(maLed, HIGH);	//on éteint la LED
}

void serialEvent()	//déclaration de la fonction d'interruption sur la voie série
{
	while ( Serial.read() != -1 );	//lit toutes les données (vide le buffer de réception)
```

```
        digitalWrite(Led, LOW);                 //on allume la LED
}
```

6. Les grandeurs analogiques

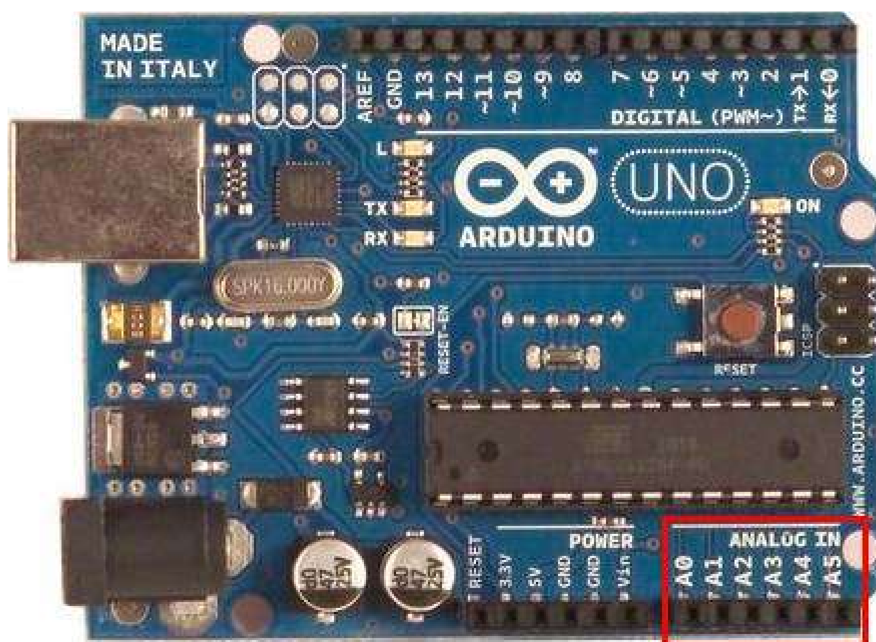
6.1. Les entrées analogiques

La carte Arduino dispose d'un CAN[1] 10 bits pour une tension analogique de 0 à +5V. Ce convertisseur, intégré dans son micro-contrôleur, est un convertisseur "à approximations successives".

La fonction analogRead() permet de lire la valeur lue sur une entrée analogique de l'Arduino. Elle prend un argument et retourne la valeur lue :

- L'argument est le numéro de l'entrée analogique à lire
- La valeur retournée (un int) sera le résultat de la conversion analogique->numérique

Sur une carte Arduino Uno, on retrouve 6 CAN. Ils se trouvent tous du même côté de la carte, là où est écrit "Analog IN" :



Ces 6 entrées analogiques sont numérotées, tout comme les entrées/sorties logiques. Par exemple, pour aller lire la valeur en sortie d'un capteur branché sur le convertisseur de la broche analogique numéro 3, on fera : valeur = analogRead(3);.

Ne confondez pas les entrées analogiques et les entrées numériques ! Elles ont en effet le même numéro pour certaines, mais selon comment on les utilise, la carte Arduino saura si la broche est analogique ou non.

La valeur retournée par la fonction est comprise entre 0 et $2^{10} - 1 = 1023$ pour une tension de 0 à +5V, soit $\frac{5}{1024} = 4{,}88$ mV par bit.

1 convertisseurs analogiques -> numérique

Voici une façon de la traduire en code :

```
int     valeurLue;      //variable stockant la valeur lue sur le CAN
float   tension;        //résultat stockant la conversion de valeurLue en Volts

void loop()
{
        valeurLue = analogRead(uneBrocheAvecUnCapteur);

         // 1ère méthode : produit en croix, ATTENTION, donne un résultat en mV !
        tension = valeurLue * 4.88;
        // 2ème méthose : formule à aspect "physique", donne un résultat en mV !
        tension = valeurLue * (5 / 1024);
}
```

Il existe également une fonction dans l'environnement Arduino qui à partir d'une valeur d'entrée, d'un intervalle d'entrée et d'un intervalle de sortie, retourne la valeur équivalente comprise entre le deuxième intervalle.

Cette fonction se nomme map(), dont voici le prototype :

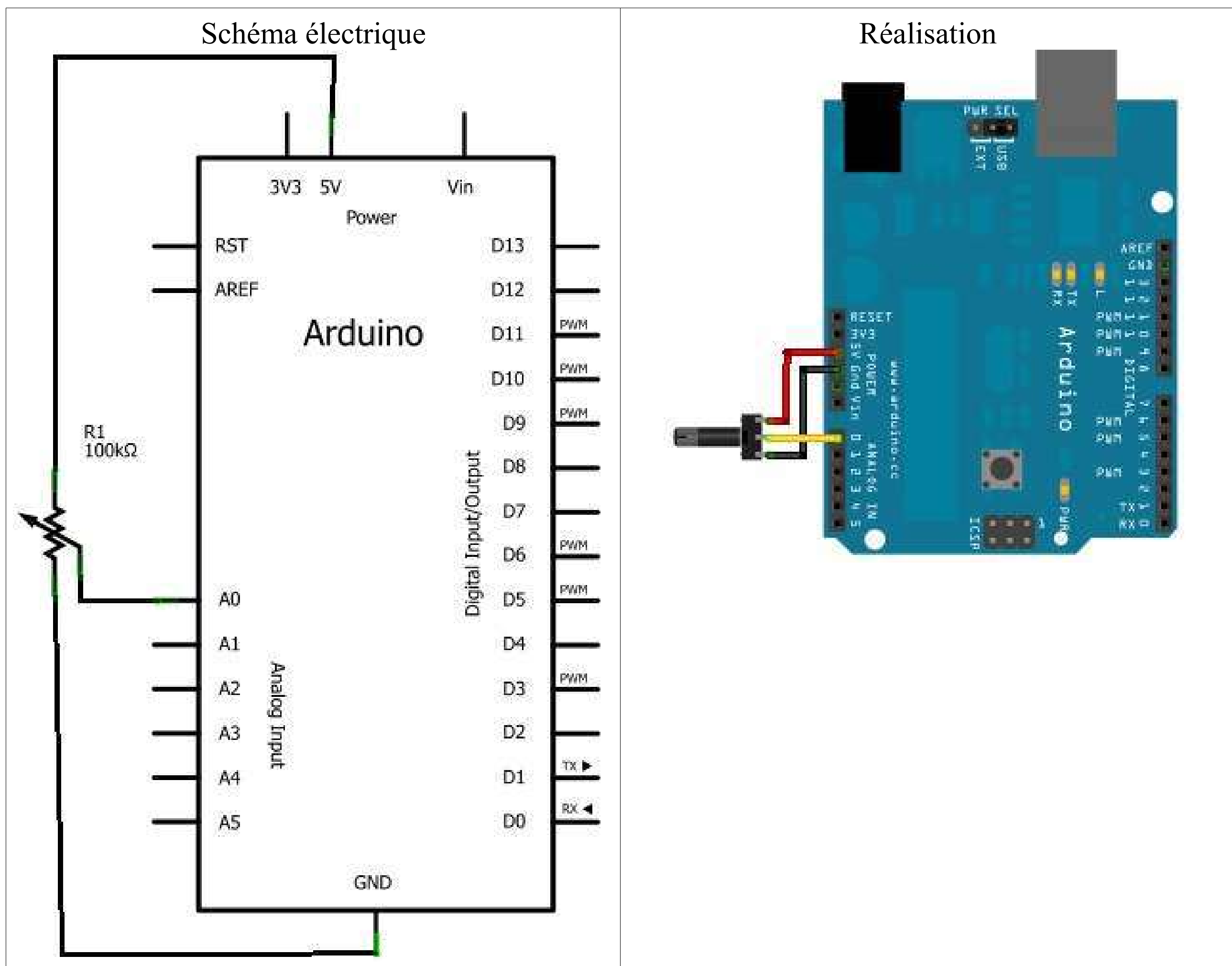
```
sortie = map(valeur_d_entree,
        valeur_extreme_basse_d_entree,
        valeur_extreme_haute_d_entree,
        valeur_extreme_basse_de_sortie,
        valeur_extreme_haute_de_sortie
        );
```

D'après l'exemple précédent, on écrira donc :

```
// map attend des valeurs entières : on utilise des mV pour être plus précis
tension = map(valeurLue, 0, 1023, 0, 5000);     //conversion de la valeur lue en tension en mV
tension = tension / 1000;                       //conversion des mV en V
```

Exemple : lecture de la valeur d'un potentiomètre linéaires (résistance variable dont la tension à ses bornes évolue de manière proportionnelle au déplacement du curseur).

Pour cela, il suffit de raccorder les alimentations sur les bornes extrêmes du potentiomètre, puis de relier la broche du milieu sur une entrée analogique de la carte Arduino :



Une fois le raccordement fait, un petit programme pour va effectuer une mesure de la tension obtenue sur le potentiomètre, puis envoyer la valeur lue sur la liaison série :

```
const int       potar = 0;      // le potentiomètre, branché sur la broche analogique 0

int             valeurLue;      //variable pour stocker la valeur lue après conversion
float           tension;        //on convertit cette valeur en une tension

void setup()
{
        //on se contente de démarrer la liaison série
        Serial.begin(9600);
}

void loop()
{
        //on convertit en nombre binaire la tension lue en sortie du potentiomètre
        valeurLue = analogRead(potar);
        //on traduit la valeur brute en tension (produit en croix)
        tension = valeurLue * 5.0 / 1024;

        //on affiche la valeur lue sur la liaison série
        Serial.print("valeurLue = ");
```

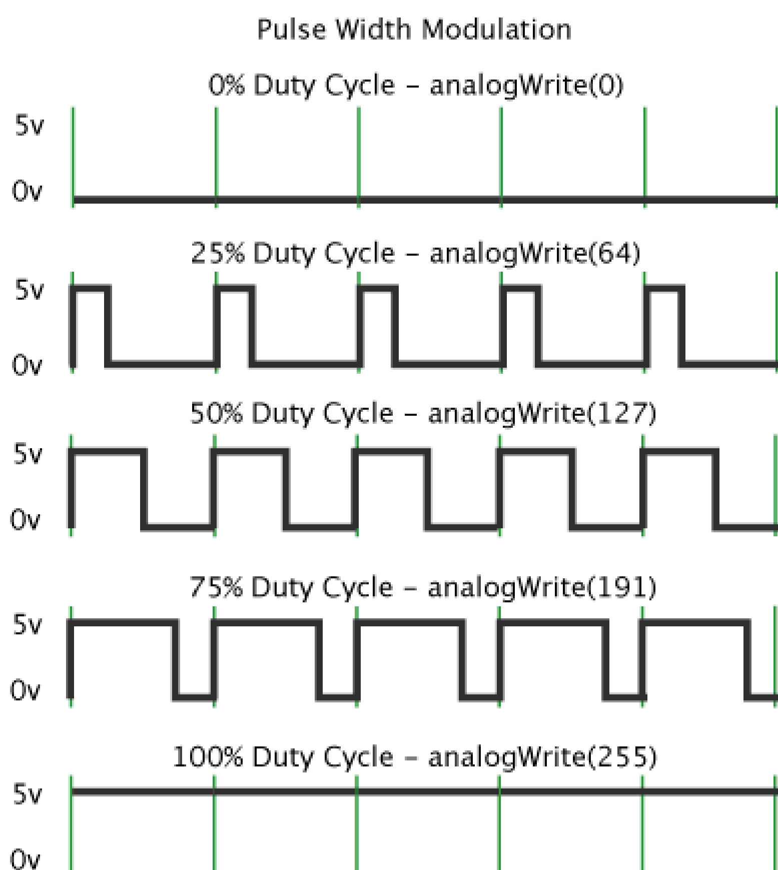
```
	Serial.println(valeurLue);

	//on affiche la tension calculée
	Serial.print("Tension = ");
	Serial.print(tension,2);
	Serial.println(" V");

	Serial.println();	//on saute une ligne entre deux affichages
	delay(500);	//on attend une demi-seconde pour que l'affichage ne soit pas trop rapide
}
```

6.2. Les sorties "analogiques"

La conversion A->N permet de transformer une grandeur analogique non-utilisable directement par un système à base numérique en une donnée utilisable pour une application numérique. Quant à la conversion opposée, conversion N->A, les applications sont différentes : par exemple commander une, ou plusieurs, LED.

Pour cela, on va utiliser ce que peut fournir la carte Arduino : la PWM[2].

La PWM est un signal numérique qui, à une fréquence donnée, a un rapport cyclique qui varie avec le temps suivant "les ordres qu'elle reçoit" :



La carte Arduino dispose de 6 broches qui soint compatibles avec la génération d'une PWM. Elles

2 Pulse Width Modulation (en français : Modulation à Largeur d'Impulsion ou MLI).

sont repérées par le symbole tilde ~ . Voici les broches générant une PWM : 3, 5, 6, 9, 10 et 11.

La fonction analogWrite() est une fonction pour utiliser la PWM qui prend deux arguments :

- Le premier est le numéro de la broche où l'on veut générer la PWM
- Le second argument représente la valeur du rapport cyclique à appliquer. Cette valeur ne s'exprime pas en pourcentage, mais avec un nombre entier compris entre 0 et 255.

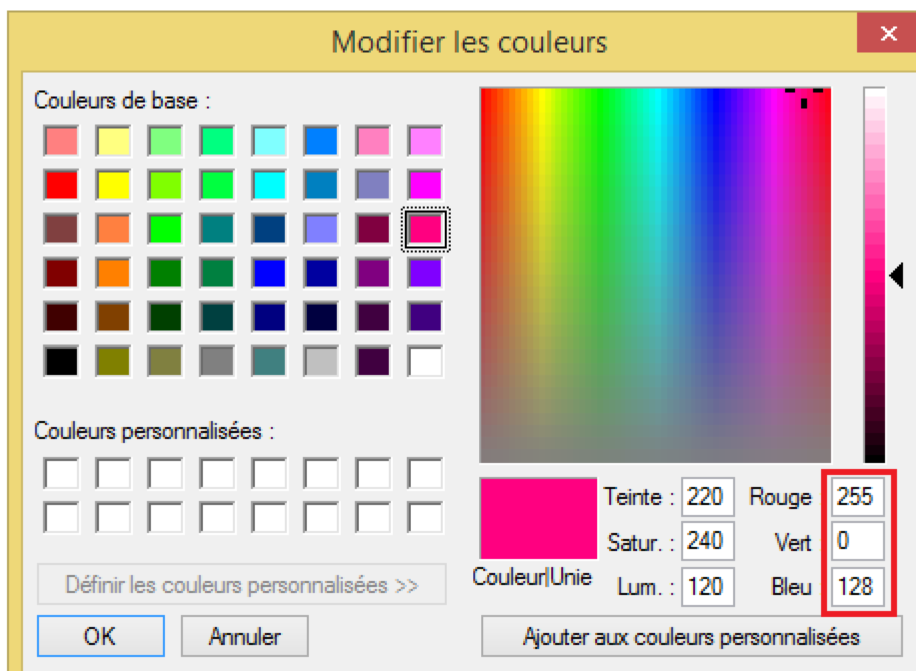
Voilà un petit exemple de code illustrant ceci :

```
const int	sortieAnalogique = 6;		//une sortie analogique sur la broche 6
void setup()
{
	pinMode(sortieAnalogique, OUTPUT);
}
void loop()
{
	analogWrite(sortieAnalogique, 107);		//on met un rapport cyclique de 107/255 = 42 %
}
```

Exemple : mixage de couleurs d'une LED RGB ou RVB.

Cette LED est composée de trois LED de couleurs primaires : Le rouge Le vert Le bleu. À partir de ces trois couleurs, il est possible de créer n'importe quelle autre couleur du spectre lumineux visible en mélangeant ces trois couleurs primaires entre elles.

Pour connaître les valeurs RGB d'une couleur, il suffit d'utiliser un logiciel de retouche d'image come GIMP ou paint.



Ci-contre un exemple de code coueur RGB :

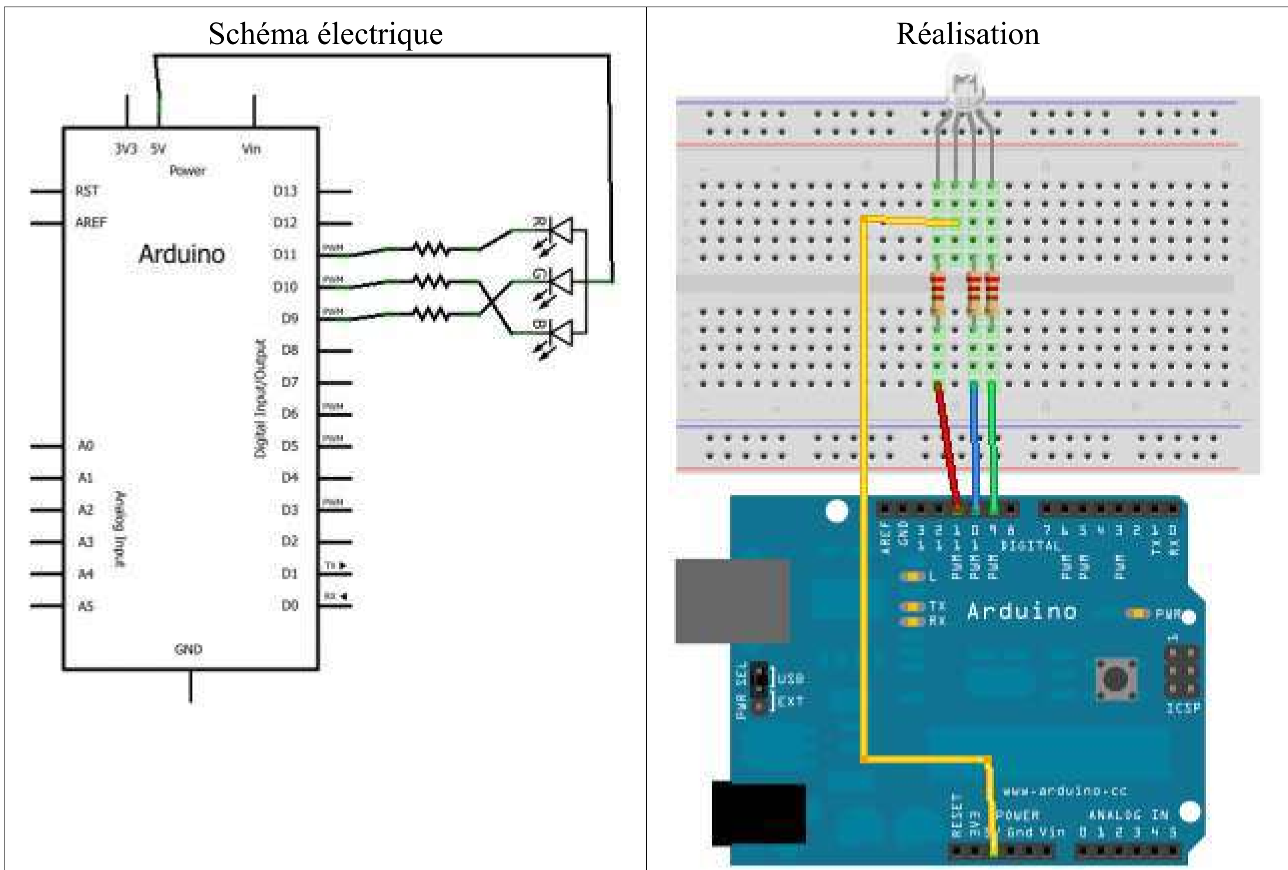
- R = 255
- V= 0
- B = 128

On utilisera ces valeurs pour faire cette teinte sur la LED RGB :

```
const int	ledRouge = 11;
const int	ledVerte =  1;
```

```
const int        ledBleue = 10;

void setup()
{
        //on déclare les broches en sorties
        pinMode(ledRouge, OUTPUT);
        pinMode(ledVerte, OUTPUT);
        pinMode(ledBleue, OUTPUT);

        //on met la valeur de chaque couleur
        //-- Les LED sont donc pilotées à l'état bas.
        //-- Ce n'est pas la durée de l'état haut qui est importante mais plutôt celle de l'état bas.
        //-- Il va donc falloir mettre la valeur "inverse" de chaque couleur sur chaque broche
        analogWrite(ledRouge, 255 - 255);
        analogWrite(ledVerte, 255 -  0);
        analogWrite(ledBleue, 255 - 128);
}

void loop()
{
        //on ne change pas la couleur donc rien à faire dans la boucle principale
}
```

Schéma électrique	Réalisation



7. Les écrans LCD[3]

Il existe un driver "LCD" pour ce type d'afficheur. Ce composant va servir à décoder un ensemble "simple" de bits pour afficher un caractère à une position précise ou exécuter des commandes comme déplacer le curseur par exemple.

Ce composant est fabriqué principalement par Hitachi et se nomme le **HC44780**. Il sert de décodeur de caractères. Ainsi, plutôt que de devoir multiplier les signaux pour commander les pixels un à un, il suffira d'envoyer des octets de commandes.

N°	Nom	Rôle
1	VSS	Masse
2	Vdd	+5V
3	V0	Réglage du contraste
4	RS	Sélection du registre (commande ou donnée)
5	R/W	Lecture ou écriture
6	E	Entrée de validation
7 à 14	D0 à D7	Bits de données
15	A	Anode du rétroéclairage (+5V)
16	K	Cathode du rétroéclairage (masse)

Normalement, pour tous les écrans LCD (non graphiques) ce brochage est le même.

Les broches utiles qu'il faudra relier à l'Arduino sont les broches 4, 5 (facultatives), 6 et les données (7 à 14 pouvant être réduite à 8 à 14) en oubliant pas l'alimentation et la broche de réglage du contraste.

Pour envoyer des données sur l'écran, il suffit de :

1. placer la broche RS à 1 ou 0 selon que l'on veut envoyer une commande
2. place sur les 8 broches de données (D0 à D7) la valeur de la donnée à afficher
3. faire une impulsion sur E d'au moins 450 ns pour indiquer à l'écran que les données sont prêtes

Ce composant possède tout le système de traitement pour afficher les caractères. Il contient dans sa mémoire le schéma d'allumage des pixels pour afficher chacun d'entre eux. Voici la table des caractères affichables :

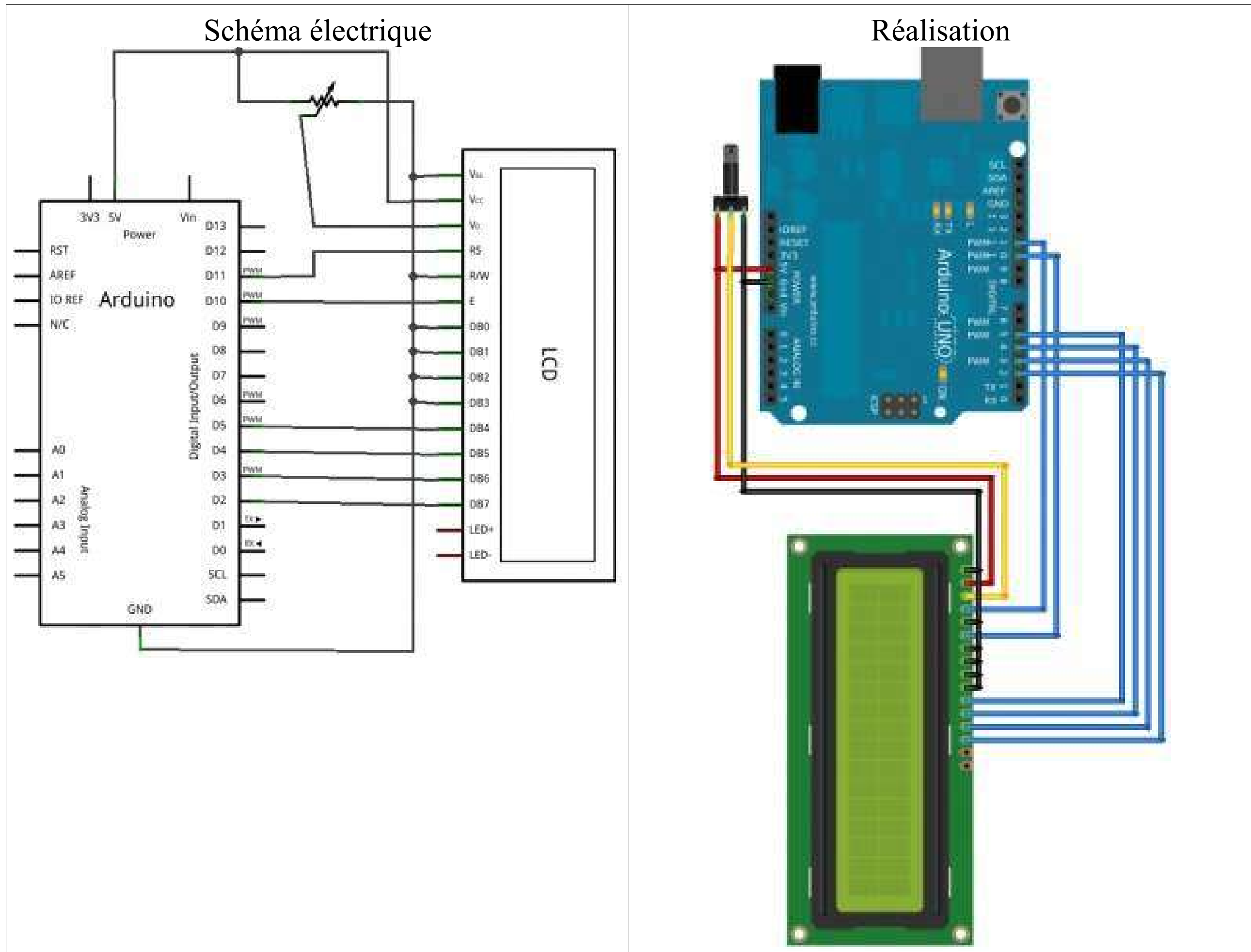
3 Liquid Crystal Display (Écran à Cristaux Liquides)

Lower 4bit \ Higher 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
××××0000			0	@	P	`	p		ー	タ	ミ	α	p
××××0001		!	1	A	Q	a	q	。	ア	チ	ム	ä	q
××××0010		"	2	B	R	b	r	「	イ	ツ	メ	β	θ
××××0011		#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
××××0100		$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
××××0101		%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
××××0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
××××0111		'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
××××1000		(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
××××1001		)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
××××1010		*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
××××1011		+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
××××1100		,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
××××1101		-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
××××1110		.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
××××1111		/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

La communication parallèle prend beaucoup de broches, il existe un mode "semi-parallèle". Ce

dernier se contente de travailler avec seulement les broches de données D4 à D7 (en plus de RS et E) et il faudra mettre les quatre autres (D0 à D3) à la masse. Il libère donc quatre broches. Dans ce mode, on fera donc deux fois le cycle "envoi des données puis impulsion sur E" pour envoyer un octet complet.

On utilisera une libraire nommée LiquidCrystal qui se chargera de gérer les timings et l'ensemble du protocole.



L'afficheur LCD utilise 6 à 10 broches de données ((D0 à D7) ou (D4 à D7) + RS + E) et deux d'alimentations (+5V et masse). La plupart des écrans possèdent aussi une entrée analogique pour régler le contraste des caractères. On branchera dessus un potentiomètre de 10 kΩ.

Les 10 broches de données peuvent être placées sur n'importe quelles entrées/sorties numériques de l'Arduino.

7.1. Afficher du texte

Pour l'intégrer la librairie "LiquidCrystal", il suffit de cliquer sur le menu "Import Library" et d'aller chercher la bonne. Une ligne #include "LiquidCrystal.h" doit apparaitre en haut de la page de code. Ensuite, il reste à dire à la carte Arduino où est branché l'écran (sur quelles broches) et quelle est la taille de ce dernier (nombre de lignes et de colonnes).

Il faudra commencer par déclarer un objet lcd, de type LiquidCrystal et qui sera global à notre

projet. Voici un exemple complet de code correspondant aux deux branchements précédents :

```
#include <LiquidCrystal.h>   //on inclut la librairie

// initialise l'écran avec les bonnes broches
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

int     heures, minutes, secondes;
char    message[16];

void setup()
{
        lcd.begin(16, 2);       // règle la taille du LCD : 16 colonnes et 2 lignes

        lcd.setCursor(4,1);     // place le curseur aux coordonnées (4,1)
        lcd.blink();            // et le fait clignoter
        lcd.print("Bonjour !");
        delay(2000);            // attente 2s
        lcd.clear();            // suppression du texte

        //changer les valeurs pour démarrer à l'heure souhaitée !
        heures = 0;
        minutes = 0;
        secondes = 0;
}

void loop()
{
        //on commence par gérer le temps qui passe...
        if ( secondes == 60 )           //une minutes est atteinte ?
        {
                secondes = 0;           //on recompte à partir de 0
                minutes++;
        }
        if ( minutes == 60 )            //une heure est atteinte ?
        {
                minutes = 0;
                heures++;
        }
        if ( heures == 24 )             //une journée est atteinte ?
        {
                heures = 0;
        }

        //met le message dans la chaine à transmettre
        sprintf(message,"Il est %2d:%2d:%2d", heures, minutes, secondes);
        lcd.home();                     //met le curseur en position (0;0) sur l'écran

        lcd.write(message);             //envoi le message sur l'écran

        delay(1000);                    //attend une seconde
```

```
        //une seconde s'écoule...
        secondes++;
}
```

NB : si jamais rien ne s'affiche, essayez de tourner votre potentiomètre de contraste.

7.2. Créer un caractère

Sur l'écran les pixels sont en réalités divisés en grille de 5x8 (5 en largeur et 8 en hauteur). C'est parce que le contrôleur de l'écran connait l'alphabet qu'il peut dessiner sur ces petites grilles les caractères et les chiffres.

Cette grille sera symbolisée en mémoire par un tableau de huit octets (type byte). Les 5 bits de poids faible de chaque octet représenteront une ligne du nouveau caractère.

Exemple : création d'un smiley, avec ses deux yeux et sa bouche.

Ce dessin se traduira en mémoire par un tableau d'octets que l'on pourra coder de la manière suivante :

```
0 0 0 0 0
X 0 0 0 X
0 0 0 0 0
0 0 0 0 0
X 0 0 0 X
0 X X X 0
0 0 0 0 0
0 0 0 0 0
```

```
byte smiley[8] = {
        B00000,
        B10001,
        B00000,
        B00000,
        B10001,
        B01110,
        B00000,
        B00000
        };
```

Une fois que le caractère est créé, il faut l'envoyer à l'écran à l'aidede la fonction createChar(), pour que ce dernier puisse le connaitre, avant toute communication avec l'écran. Cette fonction prend deux paramètres : "l'adresse" du caractère dans la mémoire de l'écran (de 0 à 7) et le tableau de byte représentant le caractère.

Ensuite, l'étape de départ de communication avec l'écran peut-être faite (le begin). Pour écrire ce nouveau caractère sur l'écran, on utilise la fonction write() en passant en paramètre l'adresse du caractère à afficher.

Remarque : cette fonction surchareg la fonction write() qui existe dans une librairie standard et qui prend un pointeur sur un char. Il faut donc faire un cast avec le type "uint8_t".

7.3. Défilement de texte

Dans un premier temps, on va faire appel aux fonctions scrollDisplayRight() et scrollDisplayLeft() pour déplacer le texte d'un carré vers la droite ou vers la gauche. S'il y a du texte sur chacune des lignes avant de faire appel aux fonctions, c'est le texte de chaque ligne qui sera déplacé par la fonction.

Exemple : défilement d'un smiley en appuyant sur deux petits boutons poussoirs.

```
#include <LiquidCrystal.h>   //on inclut la librairie
```

```
//les branchements
const int       boutonGauche = 11;  //le bouton de gauche
const int       boutonDroite = 12;  //le bouton de droite

// initialise l'écran avec les bonnes broches
LiquidCrystal lcd(8,9,4,5,6,7);

// le smiley
byte    smiley[8] = {
        B00000,
        B10001,
        B00000,
        B00000,
        B10001,
        B01110,
        B00000,
        };

void setup()
{
        //on attache des fonctions aux deux interruptions externes (les boutons)
        attachInterrupt(0, aDroite, RISING);
        attachInterrupt(1, aGauche, RISING);

        //règlage des entrées/sorties
        pinMode(boutonGauche, INPUT);
        pinMode(boutonDroite, INPUT);

        lcd.createChar(0, smiley);      //apprend le caractère à l'écran LCD
        lcd.begin(16, 2);               // règle la  taille du LCD
        lcd.write((uint8_t) 0);         //affiche le caractère de l'adresse 0
}

void loop()
{
        //pas besoin de loop pour le moment
}

//fonction appelée par l'interruption du premier bouton
void aGauche()
{
        lcd.scrollDisplayLeft();        //on va à gauche !
}

//fonction appelé par l'interruption du deuxième bouton
void aDroite()
{
        lcd.scrollDisplayRight();       //on va à droite !
}
```

Dans un deuxième temps, on va déplacer le texte automatiquement à l'aide des fonctions leftToRight() pour aller de la gauche vers la droite et rightToLeft() pour l'autre sens. Ensuite, il suffit d'appeler la fonction autoScroll() pour le faire défiler et noAutoScroll() pour l'arrêter.

Exemple : défilement des chiffres de 0 à 9.

```
#include <LiquidCrystal.h>   //on inclut la librairie

// initialise l'écran avec les bonnes broches
LiquidCrystal lcd(8,9,4,5,6,7);

void setup()
{
	lcd.begin(16, 2);		// règle la  taille du LCD
	lcd.setCursor(14, 0);		// place le curseur aux coordonnées (14,0)
	lcd.leftToRight();		//indique que le texte doit être déplacer vers la gauche
	lcd.autoscroll();		//rend automatique ce déplacement

	int	i=0;
	for (i=0; i<10; i++)
	{
		lcd.print(i);
		delay(1000);
	}
}

void loop()
{
}
```