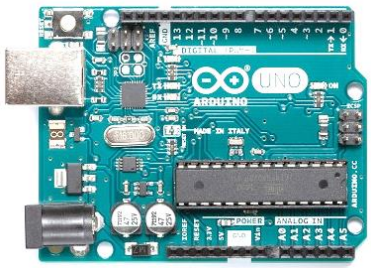
ProSysExplore

ARDUINO en pratique

Avec 10 leçons d'apprentissage

Khalid LAFKIH



Email : prosys.explore@gmail.com

Page facebook : https://www.facebook.com/ProSysExplore/

Sommaire

Avant de commencer :

Ce travail a été fait pour vous aider à apprendre une nouvelle technologie de l'électronique programmable, dans le but de vous encourager à découvrir des idées innovantes, de projets réalisables avec la carte Arduino.

Aduino est une famille de cartes électroniques, matériellement libre (Open Source), sur lesquelles se trouve un microcontrôleur (d'architecture Atmel AVR ou d'architecture ARM).

Ce microcontrôleur peut être programmé afin d'analyser et de produire des signaux électriques, de manière à effectuer des tâches très diverses qui peuvent interagir avec le monde physique, comme la domotique (le contrôle des appareils domestiques, éclairage, chauffage…), le pilotage d'un robot, le développement des systèmes embarqués…etc. Ce type de technologie, appelé souvent « physique numérique » est utilisé aujourd'hui partout dans les systèmes intelligents, à savoir les smartphones jusqu'à l'électronique automobile.

Le logiciel utilisé sur Arduino est entièrement open source et les informations de conception de matériel (schémas, plans des circuits imprimés (PCB), etc.) ont été mises à disposition sous Licence Creative Commons. En pratique, cela signifie qu'il est facile d'adapter le logiciel et le matériel à vos besoins, puis de contribuer avec ce que vous réalisez dans le projet Arduino en entier.

Le présent ouvrage traite de deux thématiques fondamentales :

- **L'électronique :** composants et fonctions.
- **La carte Arduino :** prise en main de la partie Hardware (carte électronique) + la partie Software (Environnement de développement et programmation).

Il est présenté et organisé en 10 leçons d'apprentissage avec une première partie traitant d'une introduction au sujet, ainsi que d'un certain nombre de principes et de concepts jugés suffisants pour faire une bonne entrée aux leçons.

Pré-requises :

- Connaissances de base en électronique.
- Principes fondamentaux en algorithmique et programmation.

Introduction aux systèmes de commande à base de la carte ARDUINO

SOMMAIRE

Microcontrôleur :

Dans cette première partie, nous allons nous intéresser à l'élément principal dans une carte Arduino : le microcontrôleur, à savoir sa structure matérielle générale et son principe de fonctionnement.

- **Qu'est-ce qu'un microcontrôleur ?**

Un microcontrôleur (µc) est un circuit intégré, qui regroupe sur une puce les éléments essentiels d'un ordinateur (microprocesseur, mémoires, unités périphériques et interfaces d'entrées-sorties), dans un espace très réduit, Il est généralement moins puissant en terme de rapidité ou de taille mémoire et présente un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels , Ceci en fait un composant très bon marché, parfaitement adapté pour piloter les systèmes embarquées dans de nombreux domaines d'application : Informatique, MultiMedia (téléviseurs, cartes audio, cartes vidéo…etc.), Contrôle des processus industriels(régulation, pilotage), Automobile (ABS, injection, GPS, airbag), Robotique (commande de rebots industriels), et d'autres.

La figure ci-dessous montre un microcontrôleur **ATmega 328P** qu'on trouve sur la carte Arduino :

Figure 1 : Microcontrôleur ATmega 328P (Atmel)

- **Structure matérielle d'un microcontrôleur :**

Comme j'ai indiqué dans la définition, un microcontrôleur est comparable à un mini-ordinateur, il possède les mêmes éléments que ce dernier, sauf qu'ils sont rangés dans un même boitier, ce qui lui rend simple et très pratique.

L'architecture matérielle d'un microcontrôleur est divisée en trois grandes parties :

- L'unité centrale de traitement (CPU).
- Les mémoires (RAM et ROM).

➢ Les interfaces d'entrée/sortie.

On y trouve aussi une horloge ou un oscillateur, des bus de données et un contrôleur d'interruption.

Le schéma suivant représente une structure générale simplifiée de l'architecture matérielle d'un microcontrôleur :

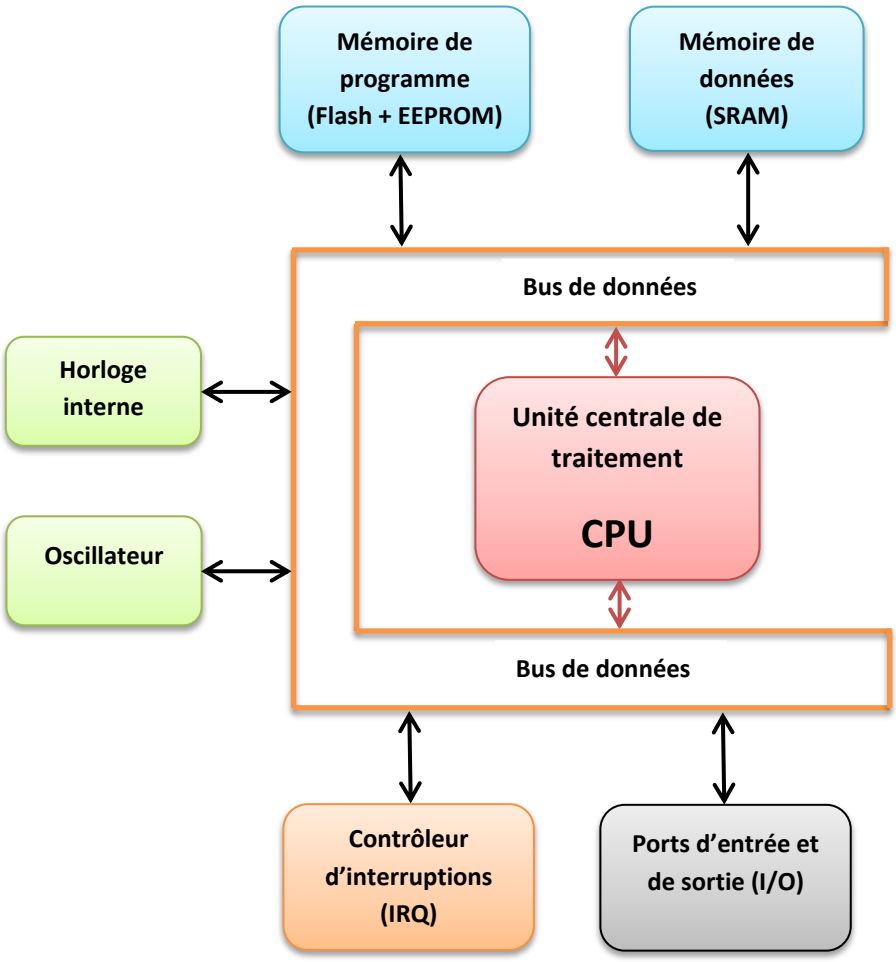


Figure 2 : Structure générale d'un microcontrôleur

Blocs principaux de la structure matérielle d'un μc :

L'unité centrale de traitement :

Appelée encore CPU (**C**entral **P**rocessing **U**nit) en anglais, c'est le cœur et l'élément le plus important dans un microcontrôleur, son rôle est l'exécution des instructions machine d'un

programme, elle peut adresser des mémoires, gérer des entrées ou sorties et réagir à des interruptions.

Le bus de données :

C'est un ensemble de fils électriques, permettant le transfert des données au sein du µc, en effet, le bus de données interconnecte les différents blocs afin de leur permettre d'échanger des données.

Les mémoires :

- **Mémoire de programme (Flash + EEPROM) :** Elle stocke le programme que le CPU doit exécuter, c'est une mémoire non volatile, c'est-à-dire qu'elle ne perde pas ses données lorsque le µc n'est plus alimenté.
- **Mémoire de données (SRAM) :** Utilisée pour gérer les résultats de calcul en cours, c'est une mémoire volatile au contraire de la mémoire programme, ça veut dire que dès l'alimentation est coupée ses données mémorisées sont perdues, pourtant, les SRAM offre un accès très rapide par rapport aux mémoires flash, ce qui présente un avantage très important.

Ports d'entrée et de sortie :

Un port d'entrée/sortie, soit numérique ou analogique (on verra par la suite la différence entre les deux), permet au µc de communiquer avec le monde extérieur, il constitue une interface à laquelle une périphérie (par exemple : une résistance, une LED, un bouton poussoir, un capteur te température, un transistor…etc.) peut être connectée.

Le contrôleur d'interruption :

Utilisé lorsqu'une interruption matérielle (en anglais **I**nterrupt **ReQ**uest ou IRQ), est déclenchée par un périphérique d'entrée/sortie du µc, à ce moment-là, ce dernier doit arrêter temporairement, l'exécution normale du programme afin de traiter un autre, appelé programme ou service d'interruption.

Ce type de programme est très utile dans des situations où on est besoin de réagir à des actions externes détectées par des capteurs par exemple. Nous verrons des exemples après.

L'horloge interne :

Elle synchronise toutes les actions de l'unité centrale de traitement.

- **Arduino et microcontrôleur ! quelle relation ?**

En regardant une carte Arduino, on peut constater tout simplement qu'elle intègre un microcontrôleur avec d'autres périphériques et composants dont nous allons découvrir en détail dans la partie consacrée à la carte Arduino.

La famille Arduino :

Les constructeurs d'Arduino fabriquent plusieurs cartes, chacune avec des capacités différentes. D'autre part, la forme, la taille et le nombre d'entrées/sorties, jouent un rôle très important dans le choix d'une carte, c'est pour cela les constructeurs ont mis un vaste choix de cartes à microcontrôleur, afin que l'utilisateur trouve le modèle qui réponde à ses besoins. En outre, ces cartes sont matériellement libres, ce qui signifie que d'autres personnes peuvent modifier et produire des dérivés de cartes Arduino qui fournissent encore plus de facteurs de forme et de nouvelles fonctionnalités.

Dans cette partie, nous allons passer en revue sur des principaux modèles de la famille, même si nous ne verrons pas tous les modèles, vous pouvez consulter le catalogue du constructeur sur le site officiel ou faire des recherches sur internet afin de découvrir d'autres types de cartes.

- **Arduino UNO :**

Basée sur un microcontrôleur ATmega 328P (Atmel), l'UNO est un excellent choix pour votre premier Arduino. Il a tout ce dont vous avez besoin pour commencer. Il possède 14 broches d'entrée / sortie numériques (dont 6 peuvent être utilisées comme sorties PWM : **P**ulse **W**idth **M**odulation ou modulation de la largeur d'impulsion en français (**MLI)**), 6 entrées analogiques, une connexion USB, une prise d'alimentation de type jack, un bouton de réinitialisation et plus encore.

La carte Uno est la première d'une série de cartes Arduino USB et le modèle de référence de la plate-forme Arduino.

Figure 3 : Arduino UNO version R3

Les différents composants et fonctionnalités de cette carte seront expliqués dans la section suivante.

Spécifications techniques de la carte UNO :

Microcontrôleur	ATmega328P
Tension de service	5V
Tension d'entrée (recommandée)	7-12V
Tension d'entrée (limites)	6-20V
Entrées/sorties numériques	14 (6 sorties commutables en MLI)
Entrées analogiques	6
Courant maxi par broche d'E/S	20 mA
Courant maxi par broche 3.3V	50 mA
Mémoire Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Chargeur d'amorçage (bootloader)	0.5 KB
Fréquence d'horloge	16 MHz
Interface	USB
Dimensions	68.6 mm* 53.4 mm
Poid	25 g

Tableau 1 : Spécifications techniques de la carte Arduino

- **Arduino Leonardo :**

Arduino Leonardo est équipée d'un microcontrôleur ATmega32u4, cette carte diffère des autres précédentes en ce que l'ATmega32u4 a intégré la communication USB, éliminant le besoin d'un processeur secondaire. Cela permet au Leonardo d'apparaître connecté sur un ordinateur, en tant que souris et clavier.

Figure 4 : Arduino Leonardo

Il peut aussi être programmé en tant que HID (**H**uman **I**nterface **D**evice), en plus, un port COM virtuel (VCP : **V**ertual **C**OM **P**ort), autre que le port matériel UART, est accessible via USB.

Spécifications techniques de la carte Leonardo :

Microcontrôleur	ATmega32u4
Tension de service	5V
Tension d'entrée (recommandée)	7-12V
Tension d'entrée (limites)	6-20V
Entrées/sorties numériques	20 (7 sorties commutables en MLI)
Entrées analogiques	12
Courant maxi. par broche d'E/S	40 mA
Courant maxi. par broche 3.3V	50 mA
Mémoire Flash	32 KB
SRAM	2.5 KB
EEPROM	1 KB
Chargeur d'amorçage (bootloader)	4 KB
Fréquence d'horloge	16 MHz
Interface	USB
Dimensions	68.6 mm* 53.3 mm
Poid	20 g

Tableau 2 : Spécifications techniques de la carte Arduino

- **Arduino Esplora :**

Arduino Esplora est une carte dérivée de l'Arduino Leonardo. Elle diffère de toutes les cartes Arduino en ce qu'il fournit un certain nombre de capteurs intégrés prêts à l'emploi. L'Esplora est conçu pour les personnes qui veulent débuter avec Arduino sans avoir à se familiariser avec l'électronique.

Cette carte dispose de sorties son et lumière intégrées, ainsi que de plusieurs capteurs d'entrée, dont un joystick, un curseur, un capteur de température, un accéléromètre, un microphone et un capteur de lumière. Il a également le potentiel d'étendre ses capacités avec deux connecteurs d'entrée/sortie Tinkerkit, et une prise pour un écran LCD couleur TFT.

Figure 5 : Arduino Esplora

Comme la carte Leonardo, l'Esplora utilise un microcontrôleur AVR Atmega32U4 avec oscillateur 16 MHz et une connexion micro USB capable d'agir en tant que client USB, comme une souris ou un clavier.

Spécifications techniques de la carte Esplora :

Microcontrôleur	ATmega32u4
Tension de service	5V
Mémoire Flash	32 KB
SRAM	2.5 KB
EEPROM	1 KB
Chargeur d'amorçage (bootloader)	4 KB
Fréquence d'horloge	16 MHz
Interface	USB

Tableau 3 : Spécifications techniques de la carte Arduino Esplora

Dimensions	164.04 mm* 60 mm
Poid	20 g

Tableau 3 (suite)

- **Arduino Mega 2560 :**

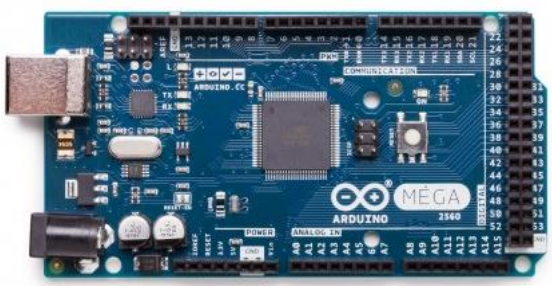


Figure 6 : Arduino Mega 2560

Basé sur un microcontrôleur ATmega2560, le MEGA 2560 est conçu pour des projets plus complexes. Avec 54 broches d'entrées/Sorties numériques, 16 entrées analogiques et un plus large espace de stockage pour les sketchs (programmes), c'est la carte recommandée pour les projets d'imprimantes 3D et de robotique.

La carte Mega 2560 est compatible avec la plupart des shields conçus pour l'Uno et les anciennes cartes Duemilanove ou Diecimila. Le Mega 2560 est une mise à jour de l'Arduino Mega, qu'il remplace.

Spécifications techniques de la carte Mega 2560 :

Microcontrôleur	ATmega2560
Tension de service	5V
Tension d'entrée (recommandée)	7-12V
Tension d'entrée (limites)	6-20V
Entrées/sorties numériques	54 (15 sorties commutables en MLI)
Entrées analogiques	16
Courant maxi. par broche d'E/S	20 mA
Courant maxi. par broche 3.3V	50 mA
Mémoire Flash	256 KB

Tableau 4 : Spécifications techniques de la carte Arduino Mega 2560

SRAM	8 KB
EEPROM	4 KB
Chargeur d'amorçage (bootloader)	8 KB
Fréquence d'horloge	16 MHz
Interface	USB
Dimensions	101.52 mm* 53.3 mm
Poid	37 g

Tableau 4 (Suite)

- **Arduino Nano :**

L'Arduino Nano est une carte compacte similaire à l'UNO.

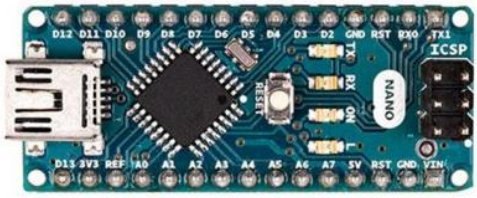


Figure 7 : Arduino Nano

Cette carte est caractérisée par sa taille qui est largement petite par rapport aux autres cartes. Equipée d'un microcontrôleur ATmega328, elle a plus ou moins la même fonctionnalité de l'ancien Arduino Duemilanove, mais dans un package différent. Il ne manque qu'une prise d'alimentation DC et fonctionne avec un câble USB Mini-B au lieu d'un câble standard.

Spécifications techniques de la carte Nano :

Microcontrôleur	ATmega328
Architecture	AVR
Tension de service	5V
Entrées/sorties numériques	22 (6 sorties commutables en MLI)

Tableau 5 : Spécifications techniques de la carte Arduino Nano

Entrées analogiques	8
Courant maxi par broche d'E/S	40 mA
Mémoire Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Chargeur d'amorçage (bootloader)	2 KB
Fréquence d'horloge	16 MHz
Interface	USB
Dimensions	18 mm* 45 mm
Poid	7 g

Tableau 5 (suite)

- **Arduino Mo :**

L'Arduino Mo est une carte à microcontrôleur 32 bits.

Le Mo représente une extension 32 bits simple mais puissante de la carte Arduino UNO. Cette carte est basée sur le microcontrôleur SAMD21 d'Atmel, doté d'un noyau ARM Cortex® Mo 32 bits.

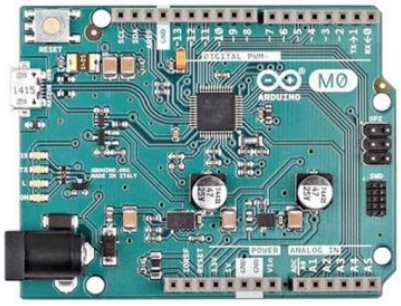


Figure 8 : Arduino Mo

Avec l'ajout de la carte Mo, la famille Arduino devient plus grande avec un nouveau membre offrant des performances accrues. La puissance de son noyau Atmel donne à cette carte une flexibilité améliorée et augmente la portée des projets que l'on peut penser et faire; de plus, cela fait du Mo un bon outil pédagogique pour apprendre le développement d'applications 32 bits.

Spécifications techniques de la carte Mo :

Microcontrôleur	ATSAMD21G18, 48 broches LQFP

Architecture	ARM Cortex-Mo+
Tension de service du µc	3.3V
Tension d'entrée générale	5-15v
Entrées/sorties numériques	20 (12 sorties commutables en MLI et UART)
Entrées/sorties analogiques	6+1 CNA
Courant maxi par broche d'E/S	7 mA
Mémoire Flash	256 KB
SRAM	32 KB
Fréquence d'horloge	48MHz
Dimensions	53 mm* 68.5 mm
Poid	21 g

Tableau 6 : Spécifications techniques de la carte Arduino Mo

- **Arduino Ethernet :**

C'est un Arduino Uno intégrant un contrôleur Ethernet TCP / IP WizNet W5100.

L'Arduino Ethernet est une carte équipée d'un microcontrôleur ATmega328. Elle se distingue des autres cartes par le fait qu'il ne dispose pas d'une puce de pilote USB-série intégrée, mais d'une interface Ethernet Wiznet. C'est la même interface que celle trouvée sur le shield Ethernet. Un lecteur de carte microSD intégré, qui peut être utilisé pour stocker des fichiers à diffuser sur le réseau, est accessible via la bibliothèque SD.

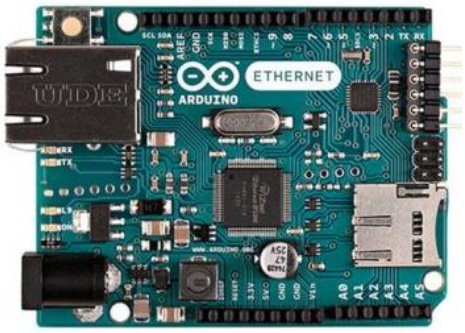


Figure 9 : Arduino Ethernet

Les broches 10, 11, 12 et 13 sont réservées pour l'interfaçage avec le module Ethernet et ne doivent pas être utilisées autrement. Cela réduit le nombre de broches disponibles à 9, 4 étant disponibles en tant que sorties PWM. Un module optionnel « Power over Ethernet » peut également être ajouté à la carte.

Spécifications techniques de la carte Ethernet :

Microcontrôleur	ATmega328
Tension de service	5V
Tension d'entrée (recommandée)	7-12V
Tension d'entrée (limites)	6-20V
Tension d'entrée Power over Ethernet (limites)	36-57V
Broches réservés	10 à 13 utilisées pour SPI (Serial Peripheral Interface) 4 utilisée pour la carte SD 2 pour l'interruption W5100
Entrées/sorties numériques	14 (4 sorties commutables en MLI)
Entrées analogiques	6
Courant maxi par broche d'E/S	40 mA
Courant maxi par broche 3.3V	50 mA
Mémoire Flash	32 KB
SRAM	2 KB
EEPROM	1 KB
Chargeur d'amorçage (bootloader)	0.5 KB
Fréquence d'horloge	16 MHz
Contrôleur Ethernet intégré TCP / IP W5100	
Prise magnétique prête à l'emploi Power Over Ethernet	
Carte Micro SD	
Dimensions	68.6 mm* 53.4 mm
Poid	28 g

Tableau 7 : Spécifications techniques de la carte Arduino Ethernet

Exploration de la carte Arduino Uno :

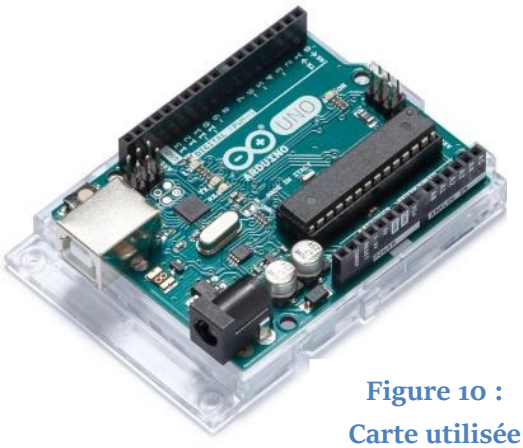


Figure 10 : Carte utilisée (Arduino Uno)

Dans cette partie, je vais commencer par vous présenter la carte qui sera utilisée dans toutes les manipulations et montages de ce livre : la carte à microcontrôleur **Arduino Uno**.

J'ai choisis cette carte car c'est la plus populaire, et la plus préférée pour débuter avec la famille Arduino.

Allons maintenant voir en détail, les principaux éléments qui la constituent.

- **Eléments principaux de la carte Uno :**

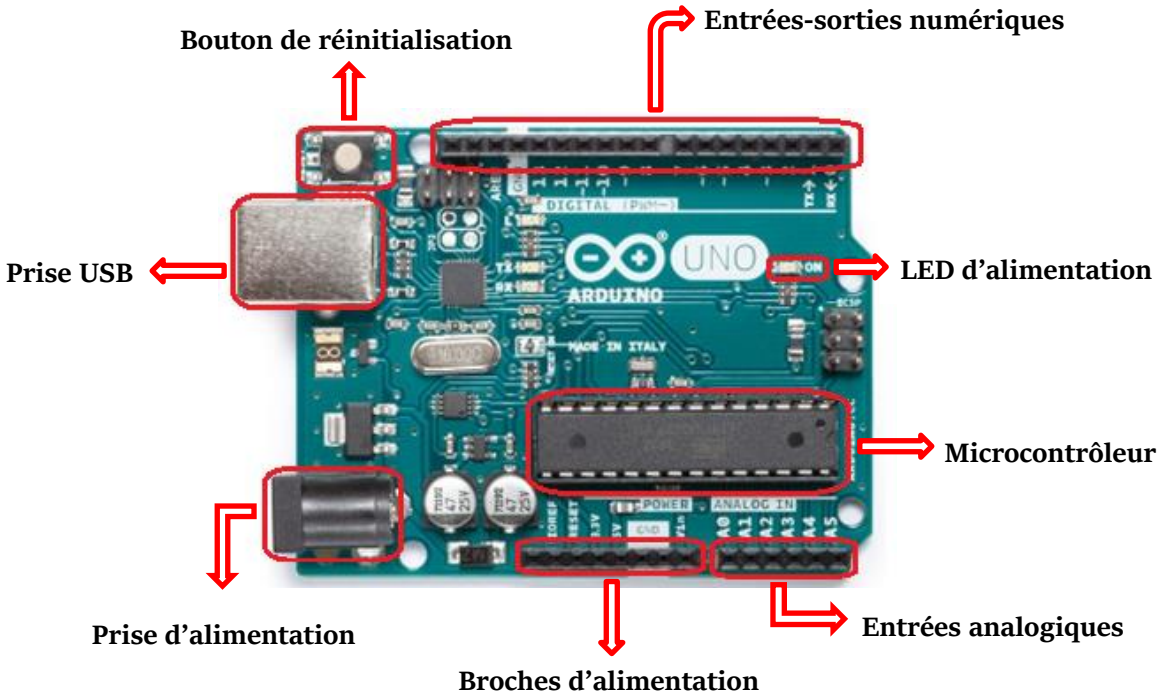


Figure 11 : Principaux éléments de la carte Uno

Ces éléments sont les plus importants à connaitre dans la carte Uno, par la suite on découvrira d'autres avec leur fonction.

Les caractéristiques principales de la carte Uno sont :

- Microcontrôleur ATmega328P.
- Une tension de service de 5V.
- 14 Entrées/sorties dont 6 sont commutable en MLI.
- 6 entrées analogiques.
- Mémoire Flash de 32Ko avec 0.5ko utilisé par le bootloader.
- Mémoire SRAM de 2 Ko.
- Mémoire EEPROM DE 1Ko.
- Fréquence d'horloge de 16MHz.
- Interface de communication USB.

L'alimentation électrique :

Il y'a deux possibilités pour alimenter la carte Uno en énergie électrique :

- Via l'interface USB qui connecte la carte à l'ordinateur, cette potion sert aussi à échanger des données entre l'ordinateur et la carte, cette interface peut fournir un courant maximal de 500 mA (largement suffisant pour réaliser des montages simples), elle est protégé contre les courants forts grâce à un polyfusible, mais malgré tout ça, il faut toujours réaliser les circuits avec plus de soin et de concentration.
- Brancher un adaptateur de tension externe ou une batterie au connecteur d'alimentation appelée prise jack.

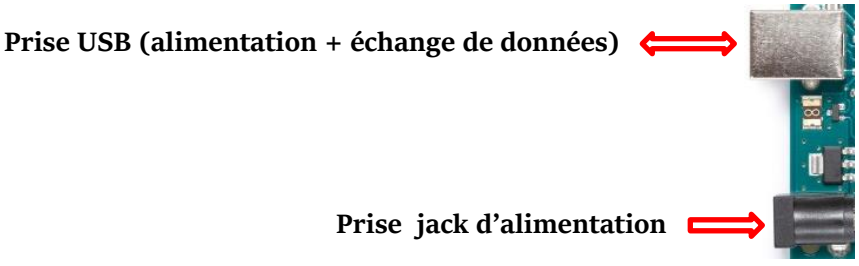


Figure 12 : Alimentation de la carte Uno

La figure ci-dessous montre les deux types de prises à utiliser avec la carte Uno :

Prise jack **Prise USB**

Figure 13 : Types de prise à utiliser avec la carte Uno

Notez bien :

- Avant de brancher une batterie ou bien un adaptateur de tension externe, il faut bien respecter les limites de tensions d'alimentation citées dans les spécifications par le constructeur, dès qu'il s'agit de tension ou de courant, il faut consulter les informations suivantes :

Tension de service	5V DC (**D**irect **C**ourant ou courant continu en français)
Tension d'entrée (recommandée)	7-12V DC
Tension d'entrée (limites)	6-20V DC
Courant maxi par broche d'E/S	20 mA
Courant maxi par broche 3.3V	50 mA

Tableau 7 : Valeurs de tensions et de courants à respecter

- Si vous utilisez un câble USB pour alimenter ou bien échanger des données avec un ordinateur, la carte sera directement liée avec ce dernier, autrement dit, la présence d'un court-circuit peut simplement affecter le port USB de l'ordinateur et endommager la carte mère. Afin d'éviter tout cela, je vous conseille d'utiliser un concentrateur

multiports USB pour brancher la carte à l'ordinateur comme le montre la figure suivante :

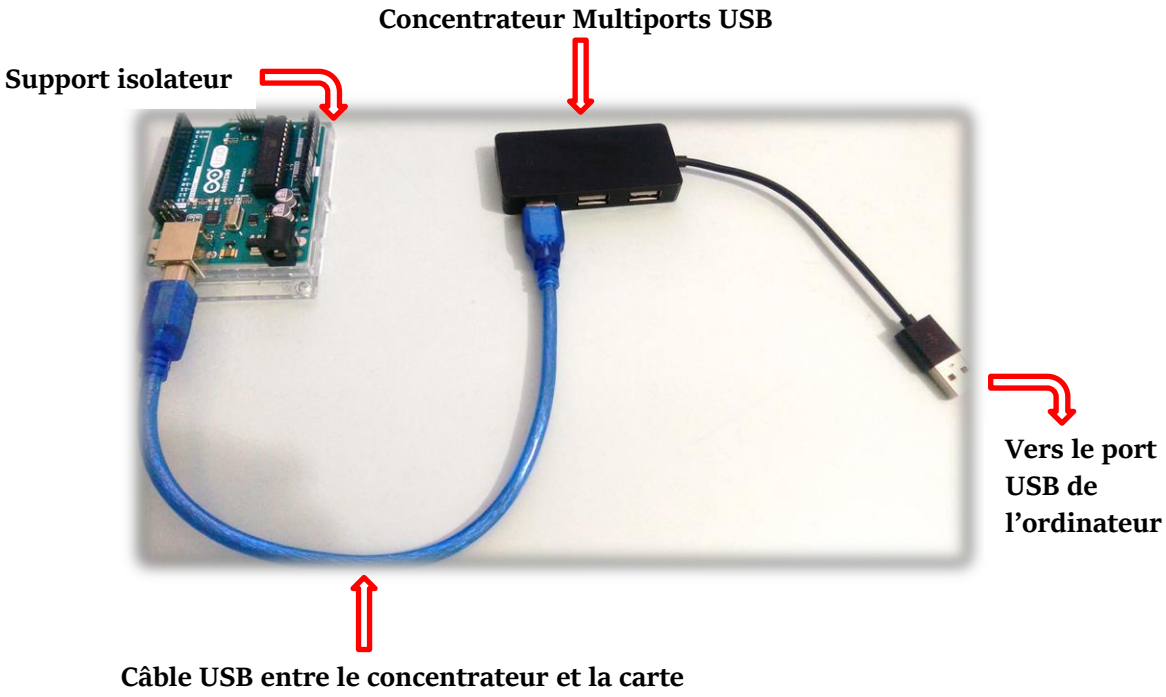


Figure 14 : Utilisation d'un concentrateur USB pour connecter la carte à l'ordinateur

Dans ce cas-là le court-circuit attaquera le concentrateur et le port USB de l'ordinateur sera protégé.

- Je profite aussi de cette occasion pour vous dire qu'il est totalement déconseillé de poser la carte directement sur un support métallique, ou une table pleine de fils dénudés, ceci provoquera un court-circuit évidement. A fin de remédier à ça, il faut utiliser des tampons en caoutchouc ou des écarteurs en les insérant sur les 4 trous forés de 3mm de diamètre sur la carte, ou bien fixer avec des vis la carte sur un support isolateur en plastique.

Port de communication USB :

En principe, un projet à base d'Arduino se devise en deux parties :

Une partie dédiée à la réalisation du montage électrique, tandis qu'une autre consacrée au développement du programme. La programmation s'effectue à travers un environnement dans lequel vous allez éditer le programme, de le compiler et le transmettre en suite via le port USB au microcontrôleur. L'opération de transfert et de chargement du programme est assurée par le chargeur d'amorçage (bootloader), c'est un petit logiciel installé dans une zone de la mémoire flash du µc.

L'ATmega328P sur l'Arduino Uno est préprogrammé avec un chargeur de d'amorçage qui vous permet de télécharger du nouveau code sans l'utilisation d'un programmateur matériel externe.

Les entrées/sorties :

Les entrées/sorties de la carte jouent le rôle d'une interface de communication avec l'extérieur et permettent d'échanger des données avec le microcontrôleur.

Leur principe de fonctionnement est illustré dans le schéma ci-dessous :

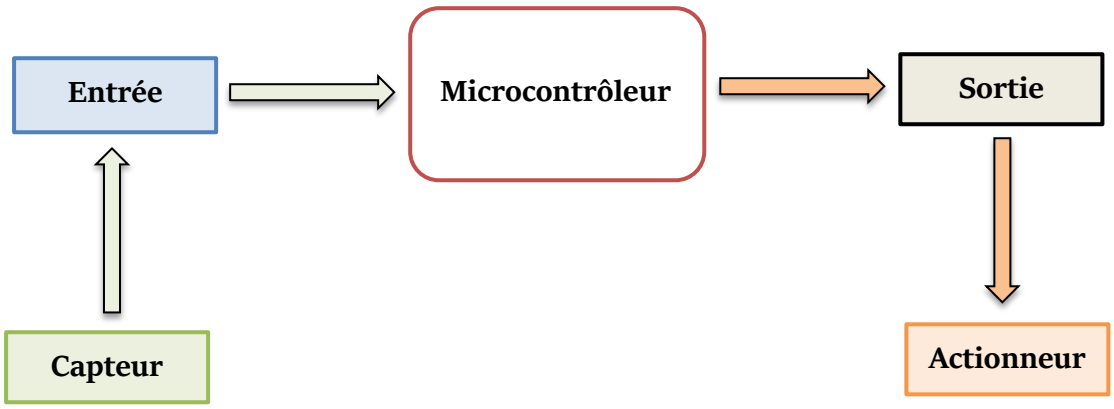


Figure 15 : Schéma fonctionnel des entrées/sorties de la carte Arduino

En général, le rôle d'une entrée de la carte Arduino est de recevoir des informations provenant d'un capteur (bouton poussoir, capteur de température, potentiomètre, résistance photo-électrique…etc.) et les transmettre au µc qui réagit, selon le programme de travail contenu dans sa mémoire pour commander la sortie. Cette dernière peut être liée à un actionneur comme un servomoteur ou bien un dispositif lumineux ou sonore.

Rappelons que la carte Arduino ne peut commander un actionneur de puissance directement, d'où la nécessité d'un pré-actionneur (relais, transistor de puissance…etc.).

Les entrées/sorties numériques :

Une entrée/sortie est une connexion physique, une "broche", sur laquelle vous pouvez venir câbler quelque chose, comme un bouton, un capteur, un voyant, etc.

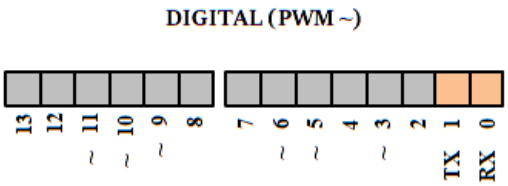


Figure 16 : Broches d'entrées/sorties numériques de la carte Uno

Une entrée / sortie numérique peut prendre l'un des trois états physiques suivant : "**haut**" (**HIGH**), "**bas**" (**LOW**) ou "**haute impédance**" (**INPUT**).

L'état "haut" (HIGH) signifie que la broche génère un signal. Cet état se traduit généralement par une tension de 5 volts en sortie de la broche avec une carte Arduino classique.

L'état "bas" (LOW) signifie que la broche ne génère pas de signal. Cet état se traduit généralement par une tension de 0 volt en sortie de la broche.

L'état "haute impédance" (INPUT) est un état un peu particulier. Dans cet état, la broche ne génère aucun signal. L'état "haute impédance" signifie que la broche est "**en lecture**" (en entrée). C'est cet état qui permet de "lire" un bouton par exemple.

La carte Uno dispose de 14 broches d'entrées/sorties numériques (figure 16). Chaque broche est programmable soit en entrée ou bien soit en sortie selon les besoins.

La numérotation des broches commence de 0 jusqu'à 13. C'est indispensable de connaitre le numéro de la broche à utiliser pour pouvoir communiquer avec lui lors de la programmation.

Les deux premières broches 0 (RX : réception) et 1 (TX : transmission), ont une deuxième fonction et sont utilisées par l'interface série de la carte.

Les entrées analogiques :

Un signal analogique est un signal qui varie dans le temps et pouvant prendre une infinité de valeurs intermédiaires comme le montre l'exemple dans le graphique ci-dessous.

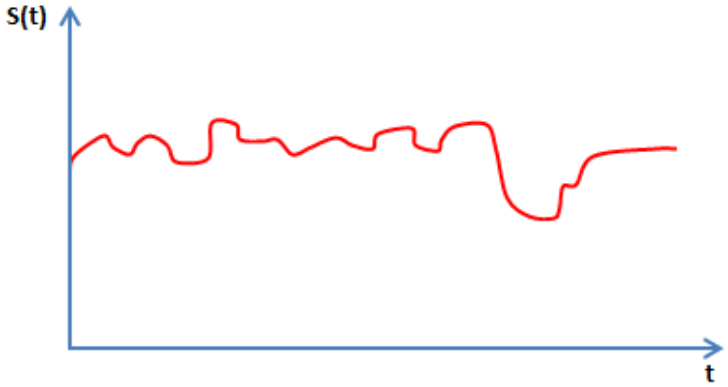


Figure 17 : Evolution d'un signal analogique en fonction du temps

On voit que son évolution est totalement différente d'un signal numérique où seule une alternative entre niveau 'HIGH' et 'LOW' est possible. Afin de traiter ce type de signaux, notre carte dispose d'entrées analogiques.

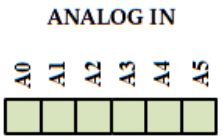


Figure 18 : Broches d'entrées analogiques de la carte Uno

En électronique numérique, on travaille avec des bits et des octets. En analogique, on travaille avec des grandeurs physiques : tension, courant, résistance, fréquence, etc.

Pour pouvoir exploiter des mesures analogiques avec le microcontrôleur, il faut convertir la mesure analogique en une grandeur numérique. C'est justement le but des convertisseurs analogique/numérique (**CAN** ou **ADC** : **A**nalog to **D**igial **C**onveter en anglais).

Un convertisseur analogique/numérique permet de mesurer une tension (valeur analogique) et de représenter cette tension au moyen d'une valeur numérique (nombre entier). L'idée est d'associer une valeur numérique pour chaque valeur analogique d'une plage de tension bien précise.

Arduino Uno possède un convertisseur analogique/numérique avec une résolution de 10 bits, il peut mesurer une tension analogique reçu sur une entrée analogique et renvoyer une valeur numérique comprise entre 0 et 1024.

Pourquoi une valeur comprise entre 0 et 1023?

Ceci est dû à la résolution du CAN($2^{10} = 1024$). La résolution est le degré auquel quelque chose peut être représenté numériquement. Plus la résolution est élevée, plus est grande la précision avec laquelle quelque chose peut être représenté. Nous mesurons la résolution en termes de nombre de bits de résolution.

Par exemple, une résolution de 1 bit n'autoriserait que deux valeurs (2^1) : zéro et un. Une résolution de 2 bits permettrait quatre valeurs (2^2) : 0, 1, 2 et 3. Si nous essayions de mesurer une plage de 0-5 volts avec une résolution de 2 bits, et que la tension mesurée était de 4 volts, notre CAN afficherait une valeur numérique de 3, car 4 volts se situent entre 3,75 et 5V. Il est plus facile d'imaginer cela avec la figure suivante:

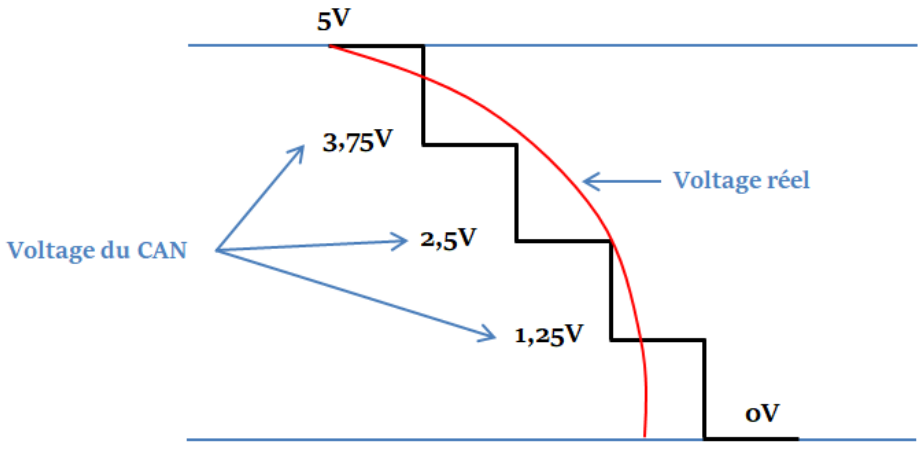


Figure 19 : Représentation d'une plage de tension analogique de 0- 5v avec un CAN de résolution 2bits

Donc, avec notre exemple CAN avec une résolution de 2 bits, il ne peut représenter la tension qu'avec quatre valeurs résultantes possibles. Si la tension d'entrée est comprise entre 0 et 1,25, le CAN renvoie le 0 numérique; si la tension est comprise entre 1,25 et 2,5, le CAN renvoie une valeur numérique de 1. Et ainsi de suite.

Remarque : Si vous délivrez à une entrée analogique une tension supérieure à 5v, vous risquez de griller ce canal ou bien d'endommager complètement le microcontrôleur de la carte. Vérifiez-vous toujours sous quelles tensions vous travaillez.

Les sorties analogiques :

Sous certains chiffres des entrées/sorties numériques se trouve un symbole (~) appelé **tilde**, indiquant que la broche est commutable en sortie analogique, il s'agit là d'une broche MLI. Les broches 3, 5, 6, 9, 10, et 11 peuvent alors, être servi comme sorties analogiques grâce à la technique de modulation de largeur d'impulsion (MLI ou PWM).

Un signal MLI (Figure 20) est un signal d'amplitude de tension et de fréquence constantes, on peut faire varier la largeur d'impulsion de ce signal en changeant la valeur d'un paramètre dit « rapport cyclique ». Plus l'impulsion est large plus la sortie reçoit d'énergie.

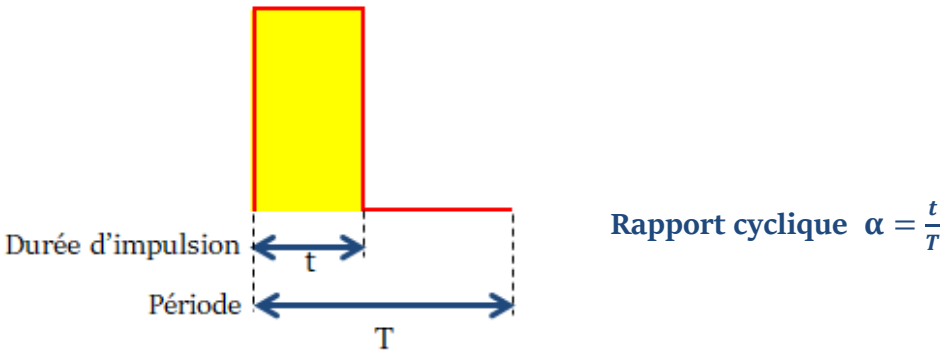


Figure 20 : Durée d'impulsion et de la période dans un signal PWM

L'interface série :

Les cartes Arduino telles que les cartes Uno, MEGA2560 et Due ont toutes un port série matériel qui utilise le port USB de la carte. Ce port permet de charger des sketchs sur la carte à l'aide d'un câble USB. Le code d'un sketch peut utiliser le même port USB/série pour communiquer avec le PC en utilisant une fenêtre du moniteur de l'interface série ou une application de traitement par exemple. Le port USB apparaît en tant que port COM virtuel sur le PC.

L'Arduino Uno n'a qu'un seul port série, car le microcontrôleur utilisé sur l'Uno ne possède qu'un seul port série intégré. L'Arduino MEGA 2560 et Arduino Due ont tous les deux 3 ports série supplémentaires.

Cette figure montre le seul port série disponible sur l'Arduino Uno encadré en rouge.

Figure 21 : Port série de la carte Uno

Ce type de communication avec votre carte Arduino peut être utile dans de nombreux cas :

- Pour entrer des données pendant l'exécution d'un sketch.
- Pour afficher des données pendant l'exécution d'un sketch.
- Pour imprimer certaines informations pendant l'exécution du sketch, afin de rechercher des erreurs.

Programmation de la carte Arduino Uno :

Arrivant à cette partie, nous allons découvrir l'environnement de développement Arduino/Genuino. Nous verrons comment télécharger et installer l'environnement de développement et nous nous intéresserons aux diverses fonctionnalités et boutons dont dispose celui-ci. En bonus, nous étudierons ensuite la structure d'un programme Arduino.

- **L'environnement de développement Arduino/Genuino :**

Pour la programmation de la carte Arduino, nous disposons d'un environnement de développement appelé IDE (Integrated Development Environment), au moyen duquel en entre directement en contact avec la carte et on charge le programme dans la mémoire du microcontrôleur. Un programme est appelé **sketch** dans le contexte Arduino.

Installation de L'IDE Arduino/Genuino :

Pour télécharger l'environnement de développement Arduino, rendez-vous sur le site officiel d'Arduino/Genuino www.arduino.cc, puis cliquez sur l'onglet SOFTWARE.

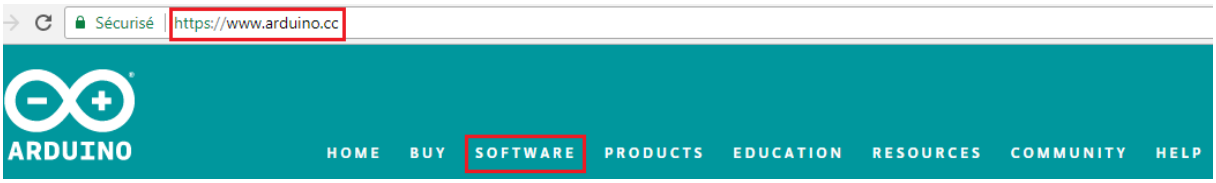


Figure 22 : Page d'accueil de arduino.cc

Sur cette page de téléchargement, vous trouverez différentes versions de l'IDE. Etants utilisateurs de Windows, La version qui nous intéresse se trouve dans l'encadré en haut de page (figure ci-dessous). Cliquez là-dessus pour obtenir la dernière version de l'IDE.Vous serez automatiquement redirigé vers une page vous proposant de soutenir le projet Arduino. Libre à vous de soutenir ou non le projet Arduino et vous pouvez cliquer sur "Just download" pour télécharger le logiciel gratuitement. Le téléchargement commence.

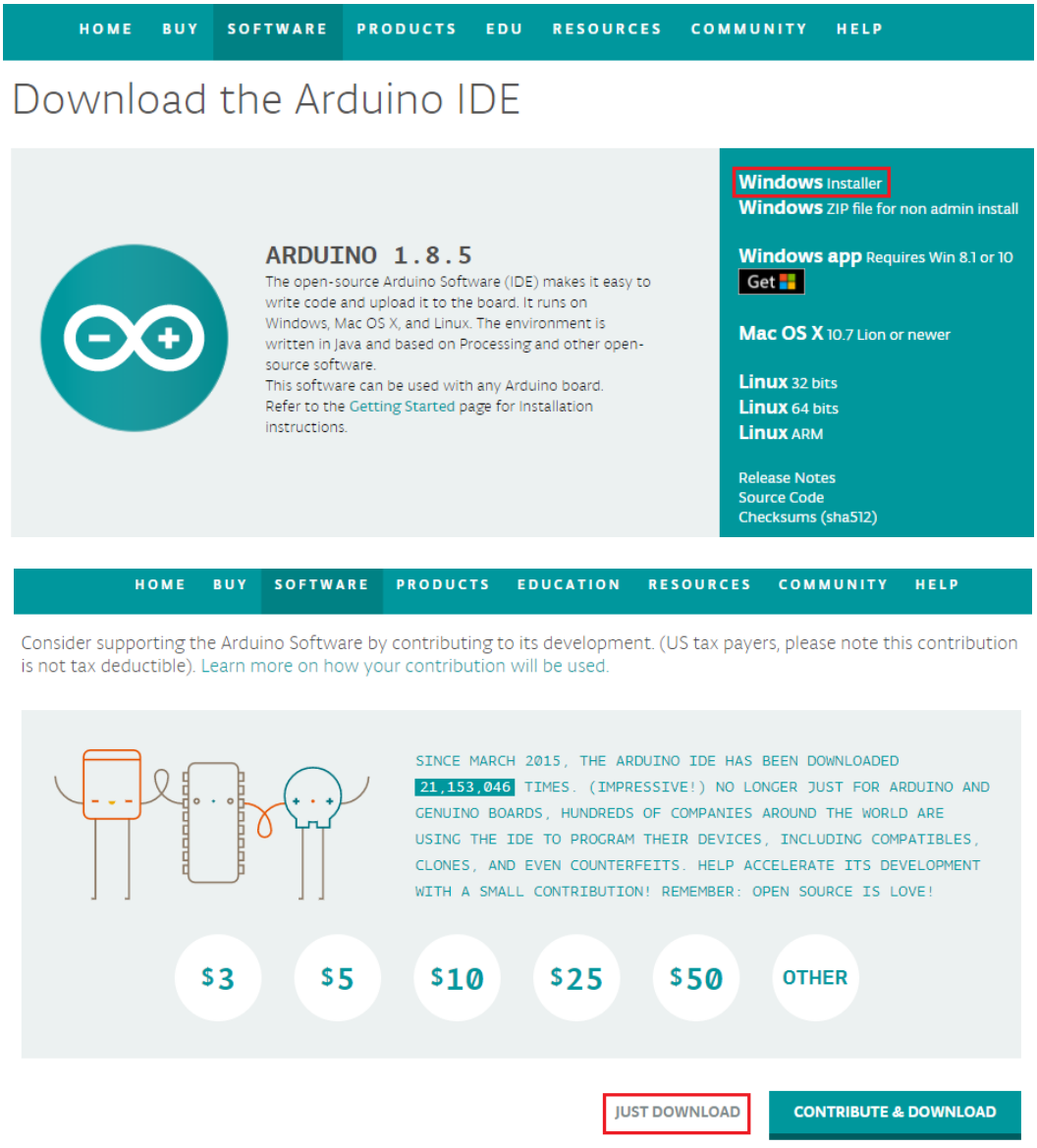


Figure 23 : Page de téléchargement de l'IDE Arduino

Une fois le téléchargement est terminé, allez-vous sur le dossier de téléchargement et vous lancez le fichier **arduino.exe,** poursuivez l'installation et autorisez le processus d'installation du pilote lorsque vous recevez un avertissement du système d'exploitation. Vous voyez alors la liste des composants qui seront installées :

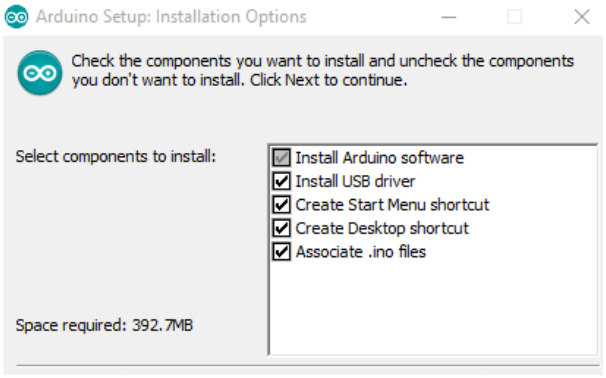


Figure 24 : Liste des composants à installer

Choisissez le répertoire d'installation (je vous conseille de garder le répertoire par défaut) :

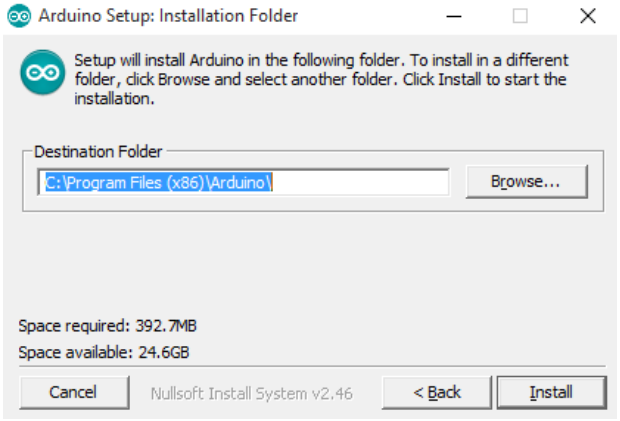


Figure 25 : Choix du chemin d'installation de l'IDE

Le processus va extraire et installer tous les fichiers requis pour exécuter correctement le logiciel Arduino IDE.

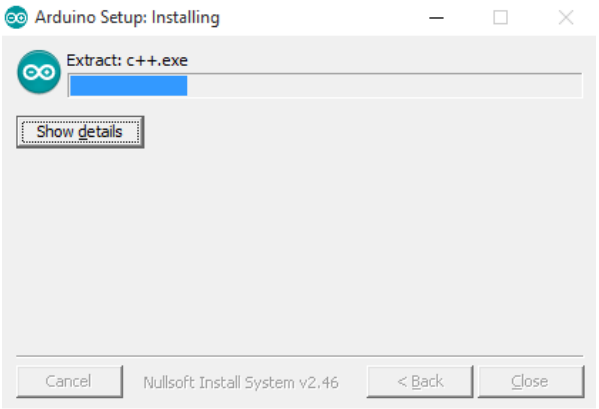


Figure 26 : Extraction des fichiers pour l'installation de l'IDE

Une fois l'installation est achevée, fermez la fenêtre.

Lorsque vous raccordez votre carte Arduino Uno avec votre ordinateur via un cable USB (figure), elle doit désormais apparaître dans le gestionnaire de périphérique dans la catégorie **Port (COM et LPT)** (figure).

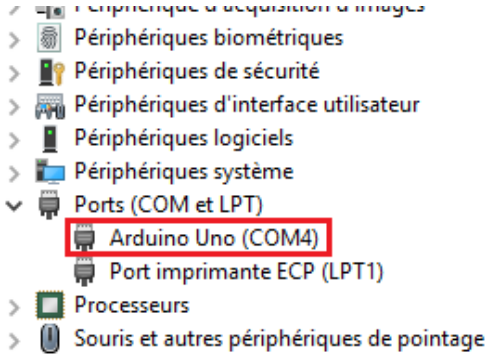


Figure 27 : Apparition de la carte Uno dans le gestionnaire de périphériques

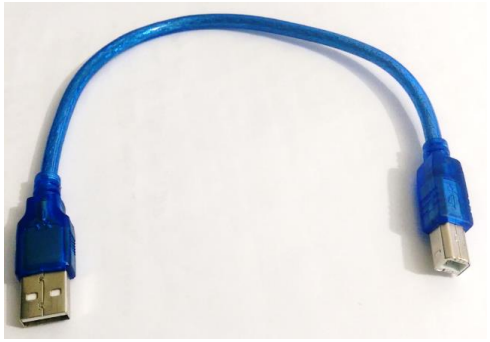


Figure 28 : Câble USB à utiliser pour relier l'ordinateur avec la carte Uno

Pour lancer l'IDE, allez dans le menu démarrer ou dans votre bureau et cliquez sur l'icône de l'IDE Arduino, vous attendez un moment, Une fois l'environnement de développement lancé, vous devriez vous retrouver devant la fenêtre ci-dessous (figure). Il s'agit de l'interface principale de l'environnement de développement Arduino.

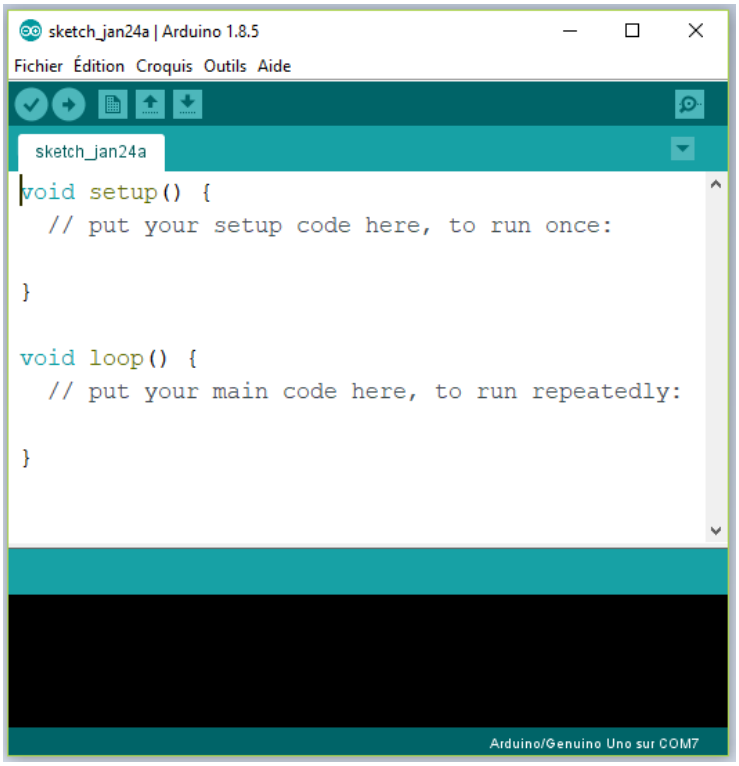


Figure 29 : Ecran principal de l'IDE d'Arduino

Prise en main de l'environnement de développement :

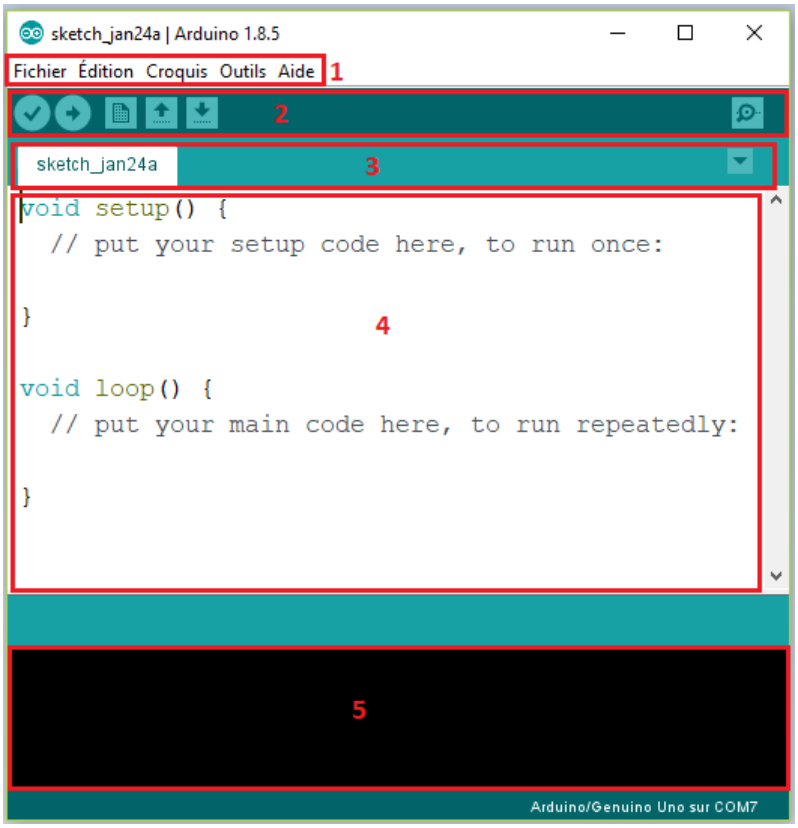


Figure 30 : Composants de l'écran principal de l'IDE

En l'observant de plus près, vous pourrez remarquer différentes zones, nous allons les passer toutes en revue, en partant du haut de la fenêtre vers le bas.

L'écran principal de l'environnement de développement Arduino est composé de 5 parties (figure) :

- 1 : Une barre de menus.
- 2 : Une barre d'îcones.
- 3 : Une barre d'onglets.
- 4 : Une zone d'édition.
- 5 : Une zone d'information et de statut.

Chacune de ces parties a une fonctionnalité bien précise que nous allons voir ensemble immédiatement.

La barre de menu :

Dans la barre de menu, vous trouverez les différents menus grâce auxquels vous pourrez appeler certaines fonctions de l'IDE.

La barre d'icônes :

La série des icônes sous la barre de menu permet de réaliser diverses actions extrêmement communes rapidement, d'un simple clic de souris.

L'icône "vérifier", permet de compiler le programme, sans le transférer sur la carte Arduino. Cet icône est très utile pour détecter les erreurs de syntaxe durant l'écriture d'un programme. Il suffit de cliquer dessus et en cas d'erreur de syntaxe, l'erreur s'affiche dans la zone d'information.

L'icône "téléverser", permet de compiler le programme, puis de le transférer dans la carte Arduino.

L'icône "nouveau", permet d'ouvrir un nouveau projet de programme Arduino.

L'icône "ouvrir", permet d'ouvrir un projet de programme Arduino existant.

L'icône "sauvegarder", permet de sauvegarder le projet de programme Arduino en cours d'édition.

L'icône "moniteur série", permet d'ouvrir le moniteur série pour communiquer avec la carte Arduino.

Les onglets :

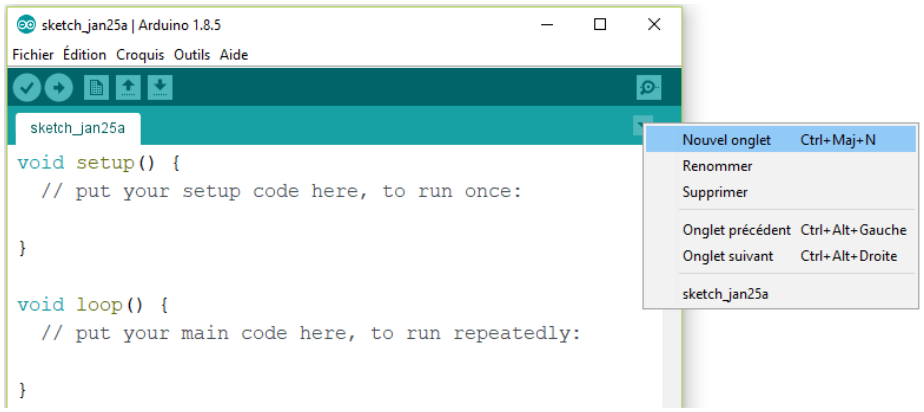


Figure 31 : Onglet de l'écran principal de l'IDE Arduino

Un projet de programme Arduino se compose d'un certain nombre de fichiers de code source. La plupart des projets Arduino se composent d'un unique fichier de code source, mais certains gros projets sont souvent découpés en plusieurs fichiers pour être plus facilement lisibles.

Chaque fichier du projet est présenté dans l'environnement de développement par un onglet. La petite flèche à droite (figure précédente) des onglets permet de gérer ceux-ci.

Il est ainsi possible d'ajouter un nouveau fichier de code source, de renommer ou supprimer un fichier existant, ou encore de naviguer dans les différents fichiers.

Le nombre de fichiers de code source peut être supérieur à ce que votre écran peut afficher. Il est alors très pratique d'utiliser les raccourcis "Onglet précédent", "Onglet suivant" et la liste d'onglet proposé par le menu flèche.

La zone d'édition :

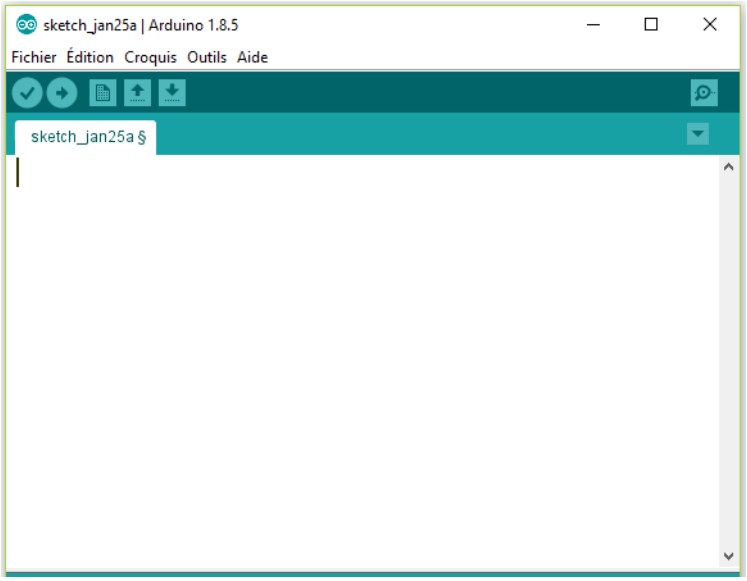


Figure 32 : Zone d'édition du programme de l'IDE

La zone d'édition qui est pour le moment encore complètement vierge (figure 32), est l'endroit où vous pouvez rédiger vos sketchs. Vous y saisissez le code source ainsi que les instructions qui conduiront le microcontrôleur à faire ce que vous voulez. La structure générale d'un sketch Arduino est représentée par l'organigramme de la figure suivante :

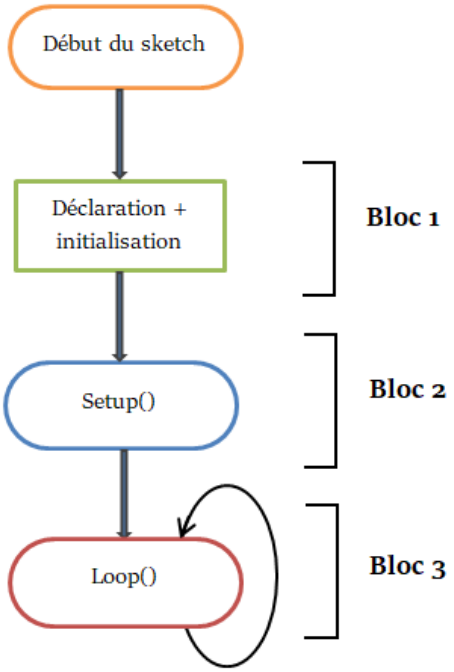


Figure 33 : Structure générale d'un sketch Arduino

Un sketch Arduino se compose de 3 blocs :

Bloc 1 (déclaration et initialisation) :

Les variables sont généralement déclarées au début d'un programme immédiatement après les commentaires d'introduction. Toutes les variables que vous utilisez dans votre code doivent être listées ici, avant les fonctions setup() et loop(). Cette déclaration permet de définir le type de donnée de la variable.

Lors de l'initialisation en revanche, la variable reçoit une valeur.

Bloc 2 (la fonction setup) :

La fonction setup permet la plupart du temps de configurer les différentes broches de la carte. On définit ainsi quelles broches doivent servir d'entrées et de sorties. On place typiquement dans le corps de cette fonction tout ce qui touche à l'initialisation du matériel (entrées, sorties, capteurs, etc.) La fonction setup () ne s'exécute qu'une seule fois, après chaque mise sous tension ou réinitialisation de la carte Arduino.

Bloc 3 (la fonction loop) :

La fonction loop() fait exactement ce que son nom suggère (boucle en français), elle boucle de façon consécutive et sans fin, permettant à votre programme de changer et de répondre.

Chacune des deux fonctions forment ensemble avec son nom un bloc d'exécution identifié par des accolades {}. Celles-ci servent d'éléments délimiteurs pour savoir où la définition de la fonction commence et où elle s'arrête. En général la structure des deux fonctions setup() et loop() dans un sketch est illustré dans l'exemple suivant :

```
void setup() {
// Une ou plusieurs instructions
//...
}
void loop() {
// Une ou plusieurs instructions
//...
}
```

La zone d'information et de statut :

Tout en bas de la fenêtre principale se trouve une zone d'information, sur fond noir.

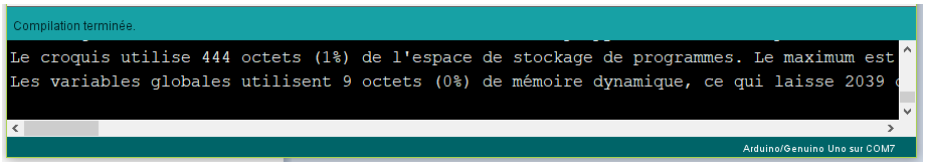


Figure 34 : Zone d'information après une compilation réussie

Cette zone d'information affiche diverses informations sur le programme, les erreurs de compilation, le transfert du programme, etc.

Figure 35 : Zone d'information indique la présence d'une erreur

En cas d'erreur, vous trouverez dans cette zone d'information, toutes les informations nécessaires pour résoudre les erreurs de syntaxes et autres problèmes de transfert.

Si une ligne d'erreur est fournie par le compilateur, celle-ci sera mise en surbrillance dans la zone d'édition (figure 35).

Transmission du sketch sur la carte Arduino :

En résumé, la transmission du sketch s'effectue en 4 étapes :

Etape 1 :

Une vérification du code du sketch est faite par l'IDE afin d'assurer que la syntaxe est correcte.

Etape 2 :

Le code est ensuite envoyé au compilateur qui transforme le code source de votre sketch Arduino en fichier binaire (au format Intel HEX) compréhensibles par le microcontrôleur de la carte : c'est le code machine.

Etape 3 :

Le chargeur d'amorçage (bootloader) transmet le fichier HEX, via USB, à la mémoir flash du microcontrôleur.

Sélection de la carte Arduino Uno :

Une fois votre sketch est achevé, vérifié et compilé avec succès, vous pourriez le transmettre au microcontrôleur de votre carte. Première des choses, vous devriez sélectionner le type de la carte avec laquelle vous travaillez (Arduino Uno dans ce cas). Pour ce faire allez-vous dans le menu **Outils** de l'IDE, choisissez 'Type de carte' et cochez la carte 'Arduino/Genuino Uno' (figure 36).

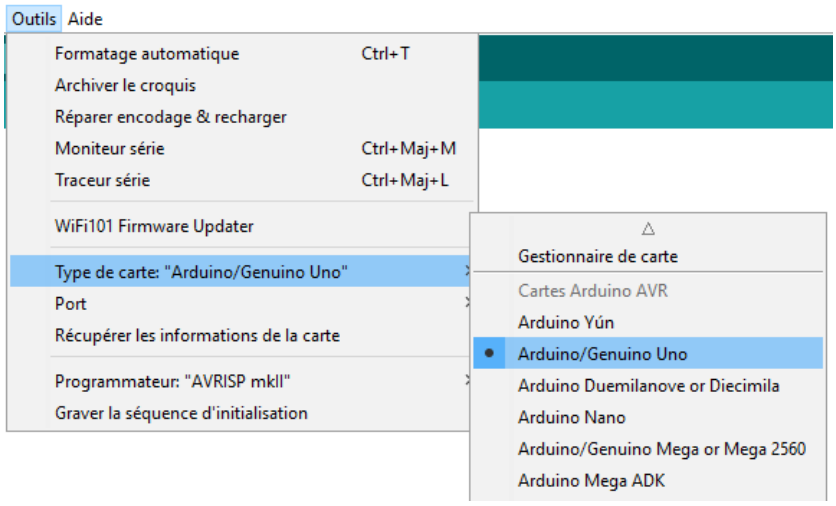


Figure 36 : Sélection de la carte Arduino Uno dans le menu 'Outils'

Sélection de port série :

Afin de pouvoir entrer en communication avec votre carte, vous devriez sélectionner le bon port COM. Vous restez toujours dans le menu Outils et vous sélectionnez 'Port' et cochez le même port COM apparaissant dans votre gestionnaire de périphériques. Dans mon cas le port COM4 sur lequel ma carte est branchée, est détecté (figure 37).

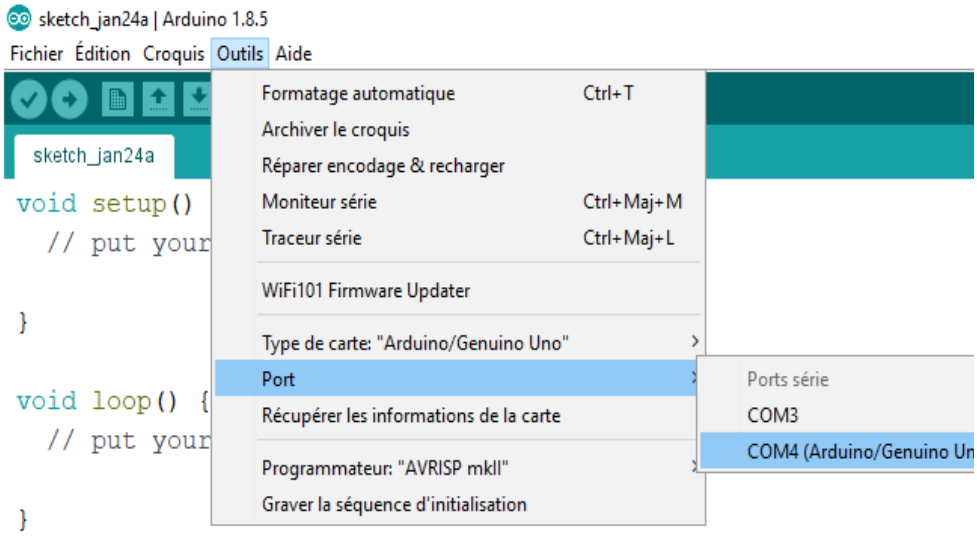


Figure 37 : Sélection du port de la carte Uno

Vous pouvez ensuite transmettre tranquillement le sketch sur la carte en cliquant sur l'icône '**téléverser**'.

Pendant le téléchargement du sketch, les trois petites LED lumineuses qui se trouvent sur la carte Uno réagissent :

Figure 38 : Les trois LED lumineuses de la carte Uno

- **LED L** : Reliée à la broche numérique 13, elle s'allume brièvement lorsque la transmission commence.
- **LED TX** : Ligne émettrice de l'interface série de la carte, elle clignote pendant la transmission.
- **LED RX :** Ligne réceptrice de l'interface série, elle clignote pendant la transmission.

La ligne émettrice (TX) est matériellement liée à la broche numériques 1 et la ligne réceptrice à la broche numérique 0.

Le moniteur série :

Le moniteur série a pour but de communiquer avec la carte Arduino connectée au PC. Celui-ci permet de transmettre et/ou recevoir des informations de la carte Arduino sous forme de texte.

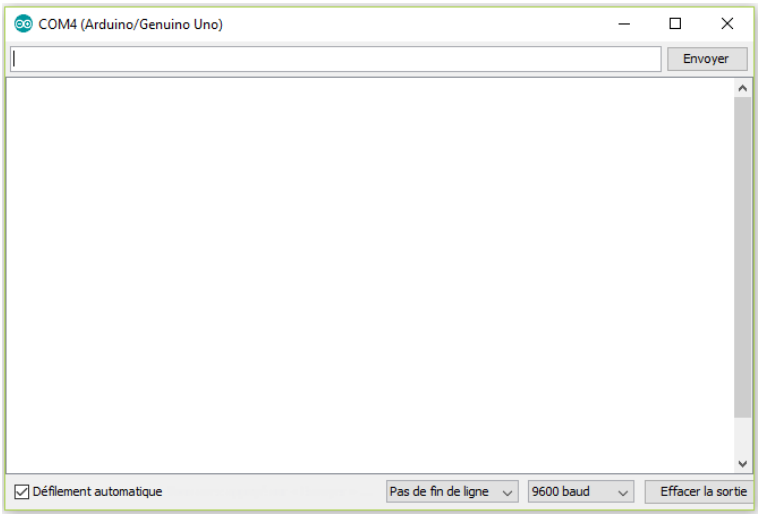


Figure 39 : Fenêtre principale du moniteur de l'interface série

Ce moniteur série est très utile lors de la conception d'un programme Arduino pour faire des tests. En affichant dans son écran des variables ou de simples chaines de caractères, il est possible de voir où son programme plante, se bloque, etc.

Dans le champ de saisi supérieur, vous pouvez entrer des commandes qui seront envoyées à la carte quand vous appuierez sur la touche **Send** et dans la partie inférieur vous pouvez, grâce à une liste déroulante à droite, de choisir la vitesse de transmission (**baud**) qui doit correspondre à la valeur que vous avez employé pour programmer le sketch. Si ces valeurs ne correspondent pas, aucune communication n'est possible.

Dans les prochaines leçons, nous allons voir des exemples de sketchs utilisant le moniteur série.

Liste des figures

Liste des tableaux

LEÇON 1

Manipulations sur les diodes électroluminescentes (LED)

Au sommaire de cette 1ère leçon :

Le premier montage « HELLO WORLD ! »

Vous avez décidé d'apprendre à pratiquer Arduino et je ne peux que vous en féliciter. Après les premières notions théoriques et l'installation de l'environnement de développement d'Arduino, nous allons découvrir ensemble dans cette première partie de leçon, le premier montage à réaliser avec une simple diode électroluminescente, apprendre à éditer ensuite, compiler et téléverser sur la carte, le code du premier sketch permettant d'effectuer la tache souhaitée.

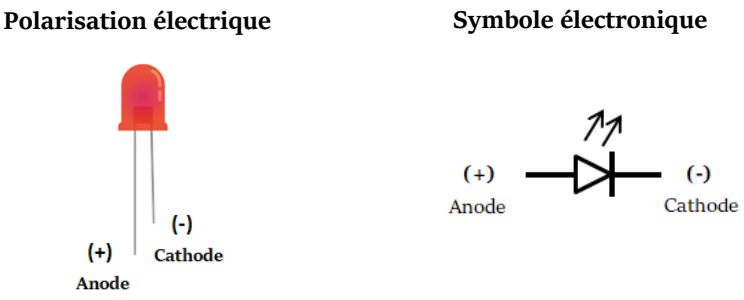
La majorité des ouvrages sur les langages de programmation, commencent par un premier programme appelé : Hello world, qui affiche sur l'écran la phrase « Hello World ! » et avec lequel les débutants découvrent ce nouveau langage et se familiarisent avec la syntaxe du programme.

Puisque notre carte Arduino ne possède pas d'écran d'affichage, nous allons essayer en premier temps, avec un programme très simple et aussi un petit montage, d'allumer la LED pour dire « HELLO WORLD ! » et faire le premier pas dans le monde Arduino.

- **La diode électroluminescente :**

Une diode électroluminescente (ou LED de l'anglais **: L**ight **E**mitting **D**iode), est un composant optoélectronique capable d'émettre de la lumière lorsqu'il est parcouru par un courant électrique. Elle produit un rayonnement monochromatique ou polychromatique non cohérent à partir de la conversion d'énergie électrique lorsqu'un courant la traverse.

Les LED existent sur le marché en plusieurs dimensions avec une diversité de couleurs que vous pouvez choisir selon les besoins de l'application. Ici nous allons utiliser une LED rouge classique (dimension 5mm) de faible puissance (< 1 w), à alimenter avec 5 Volts.



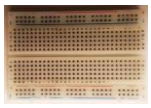
Polarisation électrique et symbole d'une LED

Pour monter une LED sur un circuit électronique, il faut bien respecter sa polarisation électrique illustrée dans la figure ci-dessus. L'anode (+) devrait être connecté à une source de tension 5v alors que la cathode (-) devrait toujours être liée à la masse (0V).

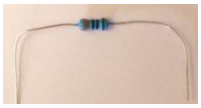
- **Le premier montage utilisant une LED :**

Matériels et composants nécessaires :


Aduino Uno

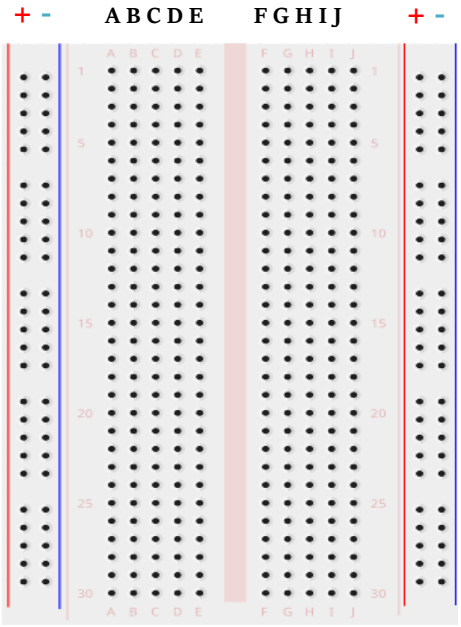

Breadboard


LED


Résistance 330 Ω


Fils de connexion

- La carte **Arduino UNO** est celle qui est utilisée dans cette série de leçons, pourtant, cela ne vous empêche pas de pratiquer sur un autre type de carte : par exemple Arduino NANO ou Arduino MEGA 2560.
- On utilise une résistance de valeur 330Ω afin de limiter le courant et protéger la LED.
- Le circuit sera réalisé sur une mini maquette d'essais appelée breadboard en anglais, à bien savoir que l'ensemble des points de connexion sur la colonne entière notée **(+)** ont une liaison continue verticale, la même chose pour la colonne notée **(-)**. Ainsi, les points de connexion sur les colonnes notées : **A B C D E**, ont une liaison horizontale, de même pour les points sur les colonnes notées **F G H I J** (voir figure à droite).



Maquette d'essais (breadboard)

- **Schéma du montage :**

C'est simple, nous allons lier la LED via la résistance R = 330 Ω avec la sortie digital 13. Comme j'ai dit précédemment sur la première partie, la carte comporte entre autres

quelques LED, dont l'une est reliée à la broche digitale 13 et possède sa propre résistance série. Nous allons donc tester, en même temps les deux LED : la L13 de la carte Arduino, et celle qu'on va brancher sur le circuit.

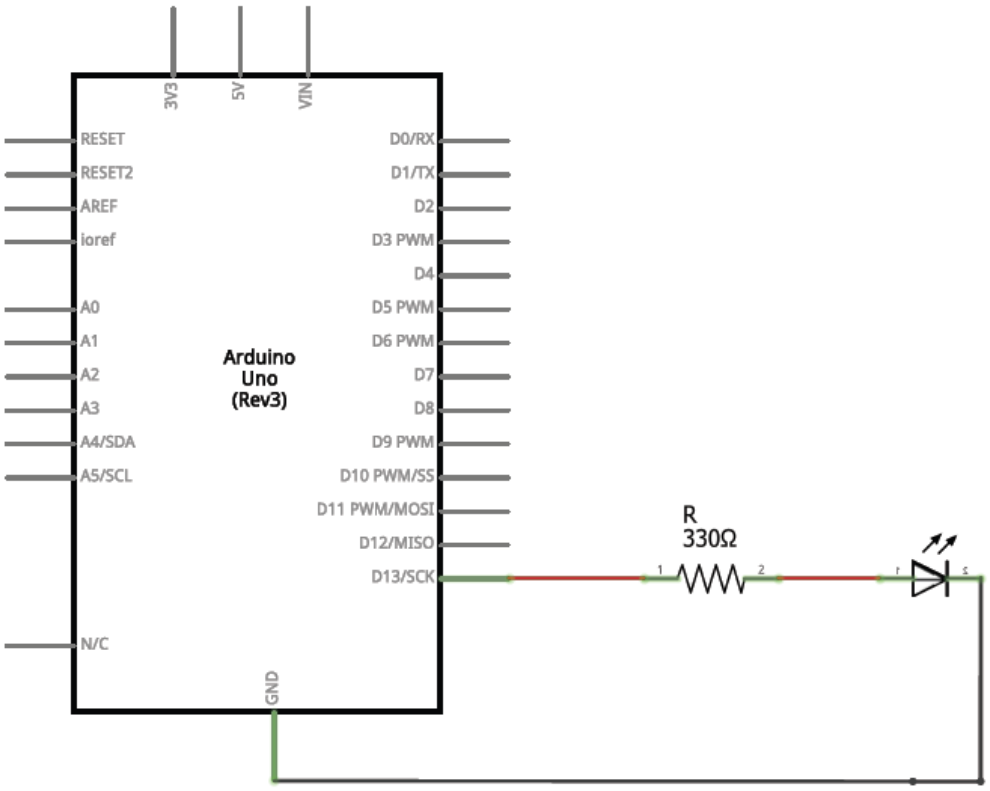
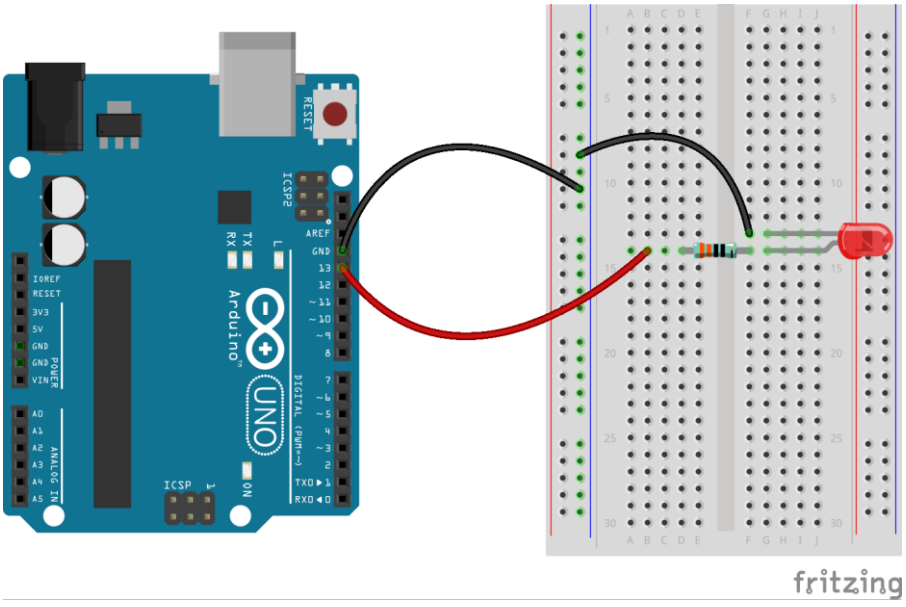


Schéma du montage

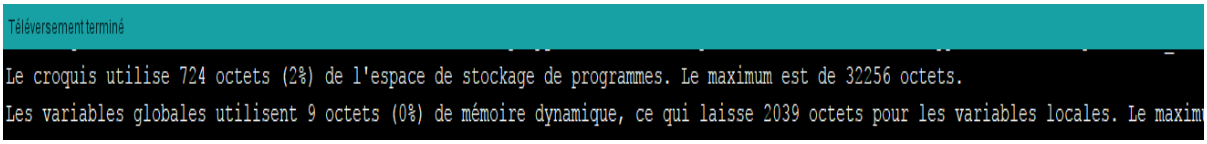


Réalisation du montage avec Fritzing

- **Code du sketch :**

```
int BrocheLed = 13;                    //Déclaration et initialisation de la broche digitale 13
void setup() {
pinMode(BrocheLed, OUTPUT);    // Programmation de la broche 13 comme sortie
}
void loop() {
digitalWrite(BrocheLed, HIGH);    // Mettre la broche 13 sur le niveau haut (5V)
}
```

Après avoir écrit ce petit programme sur l'éditeur de l'IDE Arduino, vous cliquez sur l'icône  sur la barre d'outils, afin de lancer la compilation du programme et vérifier s'il n'y a pas des erreurs de syntaxe, ensuite vous connectez votre carte Arduino à votre ordinateur via le câble USB et vous cliquez sur l'icône  afin de transmettre le sketch à la mémoire du microcontrôleur de la carte. Vous obtiendrez un message qui ressemble à celui-ci :

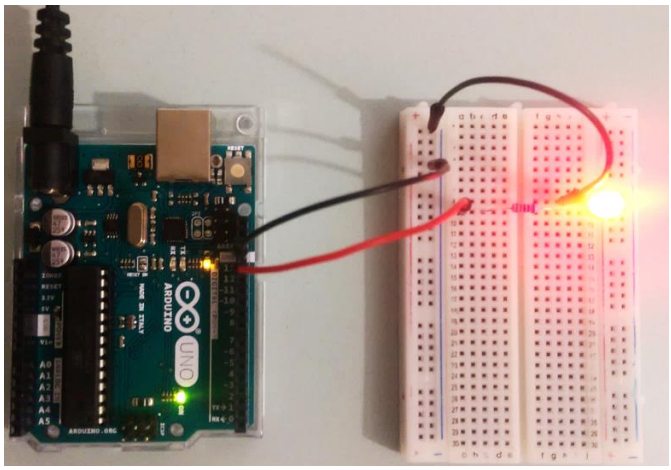


Message obtenu à la fin de la transmission du sketch

Il indique que la transmission du programme a réussi, la mémoire nécessaire y est donnée en octets, ainsi que la mémoire totale à disposition.

Une fois le téléversement du programme est terminé avec succès, vous verrez les deux LED (la L13 sur la carte et la nôtre sur le circuit) allumées qui vous saluent ! C'était votre premier montage à base d'Arduino.

Remarque : Si la transmission du sketch a échouée, vérifiez dans le menu 'Outils' de l'IDE que le bon port COM est coché et que le carte Arduino/Genuino est bien sélectionnée (voir paragraphe « **Transmission du sketch sur la carte Arduino** » de la première partie).



LED allumée pour notre premier sketch

- **Explications et analyse du sketch :**

En premier temps, nous déclarons et initialisons une variable globale qu'on a appelée BrocheLed, dont le type de donnée est un entier (int), cette variable contient donc le numéro de la broche pour la LED qu'on a liée à la sortie digitale 13 de la carte Arduino.

Remarque : dans la première ligne nous avons fait une déclaration et initialisation de la variable BrocheLed à la fois, notez bien que vous puissiez les faire séparément, déclarer premièrement, ensuite initialiser la variable, mais les deux actions ne doivent pas se suivre, sinon le compilateur déclenchera une erreur : la déclaration de la variable globale BrocheLed devrait être faite en dehors des fonctions setup et loop (boucle sans fin), tandis que l'initialisation se fait quant à elle dans la fonction setup, dans ce cas-là, le code correcte est le suivant :

```
int BrocheLed;                    //Déclaration de la variable
void setup() {
BrocheLed = 13 ;                  // Initialisation de la variable BrochLed avec la valeur 13
pinMode(BrocheLed, OUTPUT);       // Programmation de la broche 13 comme sortie
}
void loop() {
digitalWrite(BrocheLed, HIGH);    // Mettre la broche 13 sur le niveau haut (5V)
}
```

La fonction setup est appelée une seule fois pour démarrer le sketch, et la broche digitale 13 est programmée comme sortie en utilisant l'instruction pinMode, cette instruction présente deux arguments, avec le premier on désigne le nombre de la broche à programmer et avec le deuxième on choisit le mode de configuration : soit en sortie (en mettant la constante OUTPUT) ou soit en entrée (en mettant la constante INPUT).Dans notre cas on avait besoin d'une sortie pour la LED.

En fin, l'instruction digitalWrite se charge à écrire un niveau HIGH (haut = 5V) ou un niveau LOW (bas = 0V) sur une broche. Dans son premier argument on indique le numéro de la broche et sur le deuxième le niveau qu'on veut écrire sur cette broche. Dans notre cas, on avait besoin d'écrire un niveau HIGH (5V) pour allumer la LED.

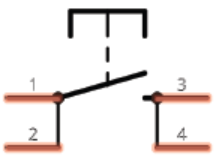
L'interrogation d'un capteur TOR : le bouton poussoir (BP)

Maintenant que nous avons une LED qui s'allume, nous allons essayer dans cette partie, d'allumer ou éteindre la LED en fonction de l'état d'un capteur TOR (**T**out **O**u **R**ien).

Le concept TOR se ramène au binaire : 0 ou 1. Cela signifie que l'information à traiter ne peut prendre que deux états (marche-arrêt). Seuls ces deux niveaux logiques sont possibles, d'où l'appellation commande tout ou rien. Un bouton poussoir, un thermostat… etc, constituent des dispositifs tout ou rien.

- **Le bouton poussoir :**

Le type du bouton poussoir que nous allons exploiter dans le montage est 'normalement ouvert' (NO) (figure ci-dessous), c'est un bouton qui est ouvert par défaut (pas de contact entre les deux broches du bouton) et quand on appuie sur lui, il se ferme et fait contact entre ces deux broches. Il existe aussi un type de bouton qui est normalement fermé (NC). Il fait la fonction inverse du type précédent.

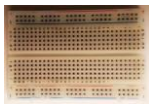


Symbole électronique d'un bouton poussoir NO

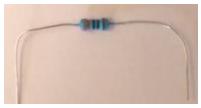
Bouton poussoir

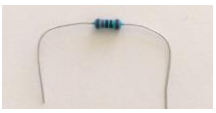
Matériels et composants nécessaires :


Aduino Uno


Breadboard


LED


Résistance 330 Ω


Résistance 10 kΩ

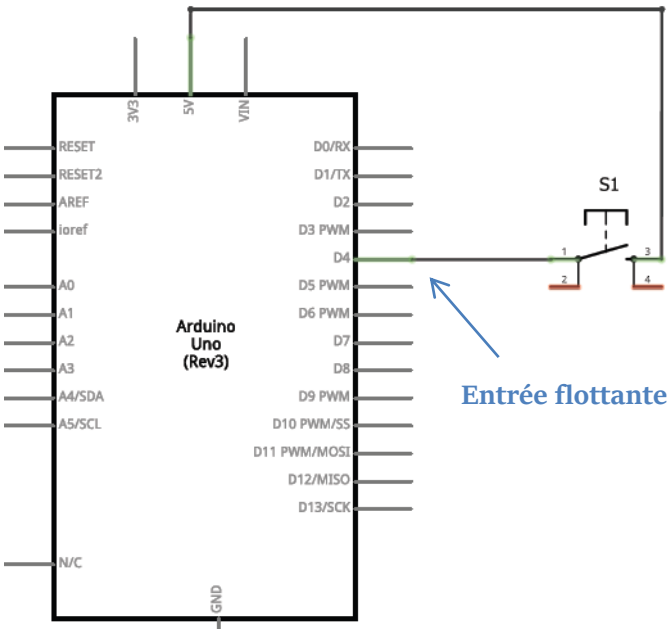

Fils de connexion


Bouton poussoir NO


Condensateur 1 µF

- **Branchement du bouton poussoir :**

Commençons par ce premier montage où nous allons brancher directement, le bouton poussoir entre l'alimentation 5V et l'entrée numérique 4.



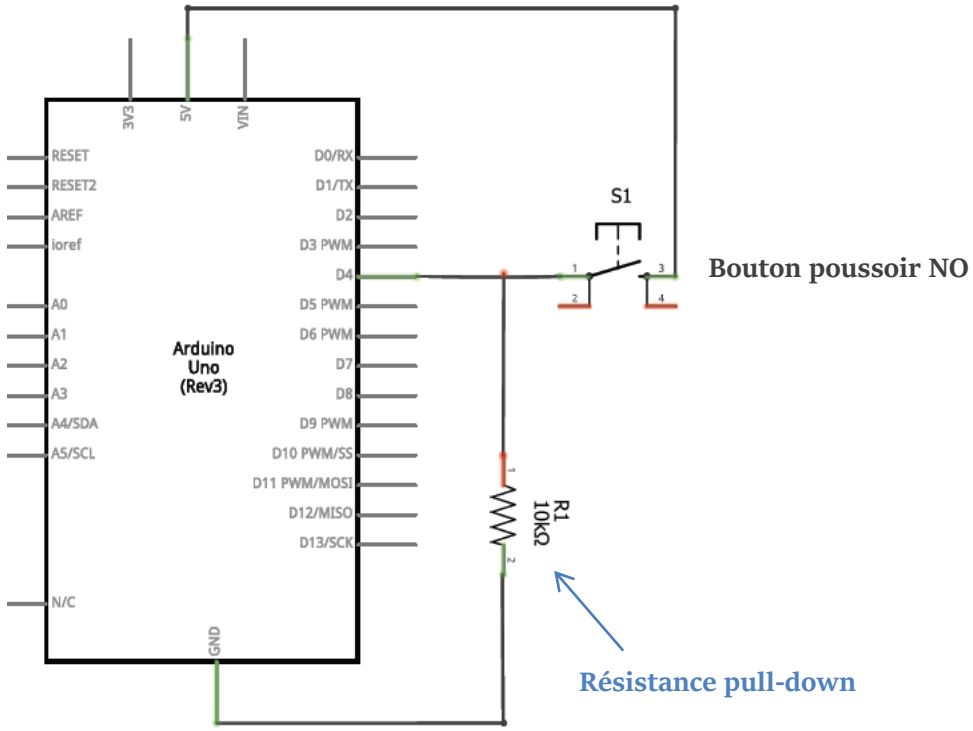
Entrée flottante d'un bouton poussoir branché sur la carte Uno

Cela ne fonctionnera pas correctement car l'entrée du bouton poussoir (broche numérique 4 dans cet exemple) est "flottante" si le bouton poussoir est non appuyé (ouvert).

Le BP est définitivement à +5V lorsqu'il est fermé , Mais n'est pas forcément à 0V lorsqu'il est ouvert. Il existe trois méthodes principales pour obtenir un fonctionnement fiable, elles sont décrites ci-dessous.

Montage avec résistance pull-down :

Le circuit ci-dessous ajoute une résistance de 10KΩ "pull-down" (la valeur exacte n'a pas d'importance, tant qu'elle n'est pas trop basse).

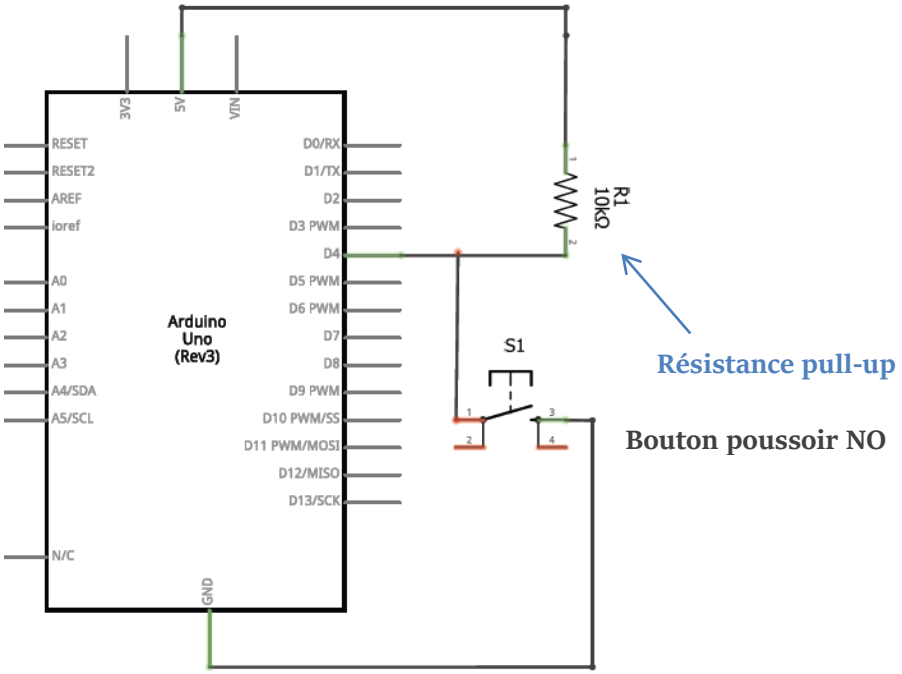


Montage avec résistance pull-down

Lorsque le bouton-poussoir est ouvert (non pressé), il n'y a pas de connexion entre les deux branches du bouton-poussoir, donc la broche est connectée à la masse (à travers la résistance pull-down) et nous lisons un état LOW. Lorsque le bouton est fermé (pressé), il fait une connexion entre ses deux jambes, reliant la broche à 5 volts, de sorte que nous lisons un état HIGH.

Montage avec résistance pull-up :

Nous pouvons également câbler le circuit dans le sens inverse, avec une résistance pull-up gardant l'entrée HIGH, et passant à LOW lorsque le bouton est enfoncé.

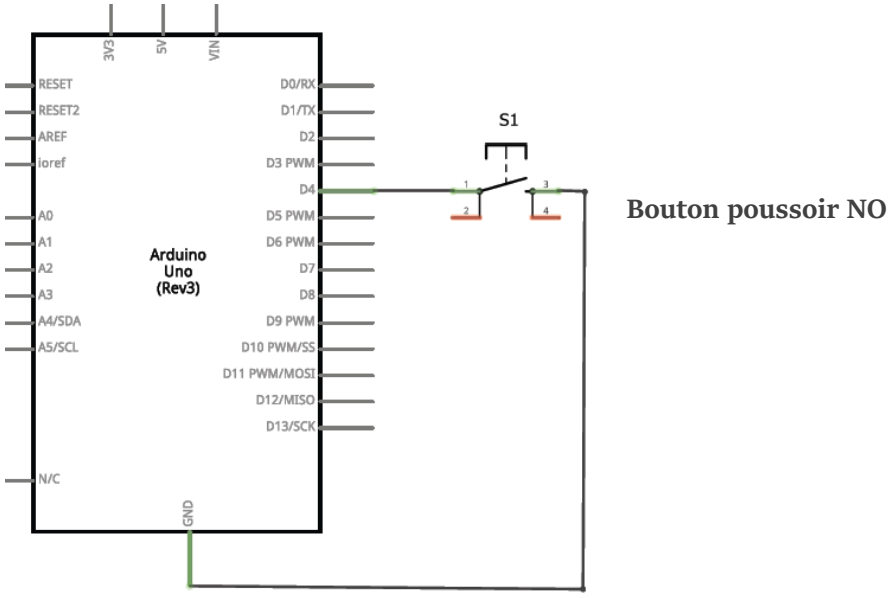


Montage avec résistance pull-up

Cette résistance tire "faiblement" le BP à + 5V, de sorte que si le BP est ouvert, il aura 5V sur lui (à travers la résistance) et ainsi enregistrera HIGH s'il n'est pas pressé, et LOW s 'il est pressé.

Montage avec résistance pull-up interne :

Le circuit ci-dessous n'a pas de résistance, mais il repose sur le "pull-up interne" que nous pouvons activer dans le code du sketch.



Montage pour l'utilisation de la résistance pull-up interne

Cette résistance pull-up interne tire "faiblement" le BP à + 5V, de sorte que si le BP est ouvert, il aura 5V (à travers la résistance interne) et donc enregistrera HIGH s'il n'est pas pressé, et LOW si pressé. Nous allons voir en suite, dans un exemple, comment activer cette résistance.

- **Schéma du montage avec résistance pull-down :**

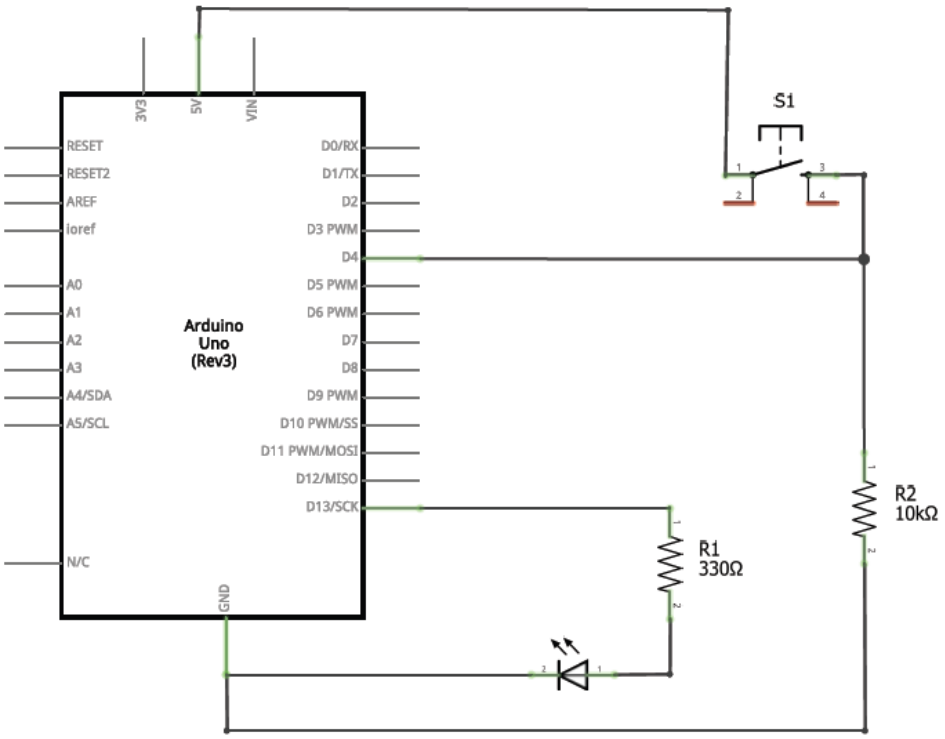
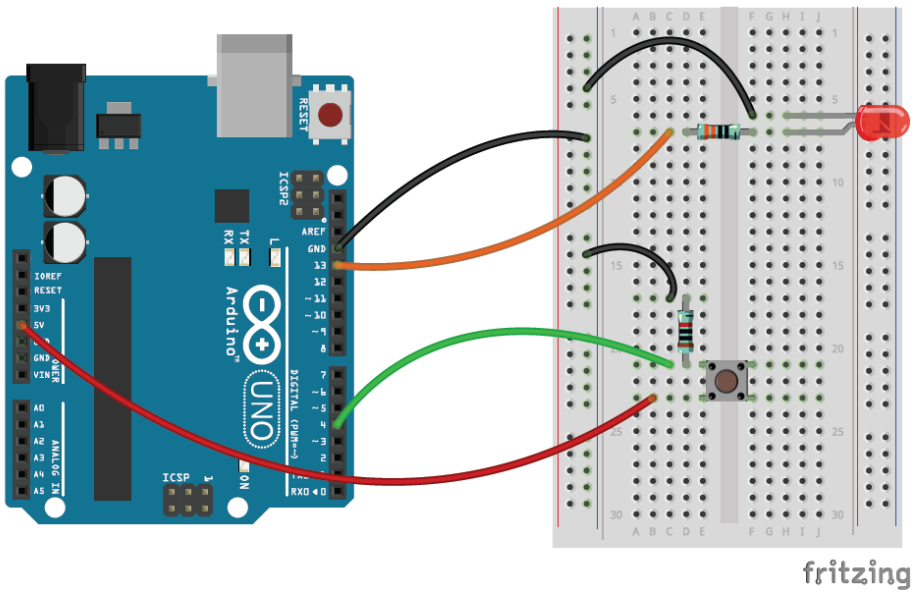


Schéma du montage



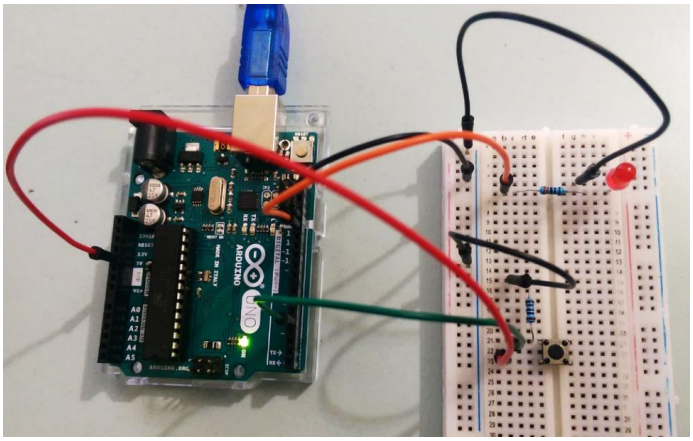
Réalisation du circuit avec fritzing

- **Code du sketch :**

```
int BrocheLed = 13;                 //Déclaration et initialisation de la broche digitale 13
int BrocheBouton = 4;               //Déclaration et initialisation de la  broche digitale 4
void setup() {
pinMode(BrocheLed, OUTPUT);     // Programmation de la broche 13 comme sortie
pinMode(BrocheBouton, INPUT);  // Programmation de la broche 4 comme entrée
}
void loop() {
if (digitalRead(BrocheBouton) == HIGH) // Si l'entrée du BP est à l'état haut (BP pressé)
digitalWrite(BrocheLed, HIGH);      // Mettre la broche 13 sur le niveau haut (5V)
else                                // Sinon (l'entrée du BP est à l'état haut (BP relâché))
digitalWrite(BrocheLed, LOW);     // Mettre la broche 13 sur le  niveau bas (0V)
}
```

- **Explications et analyse du sketch :**

Après avoir déclaré et initialisé les deux broches numériques : 'Brochled' pour la sortie 13 liée avec la LED et 'BrocheBouton' pour l'entrée 4 liée au bouton poussoir, dans la fonction setup nous avons configuré la broche 13 comme sortie et la broche 4 comme entrée. Au sein de la fonction loop nous avons utilisé la fonction digitalRead (permettant de lire et nous renvoyer l'état (HIGH ou LOW) d'une entrée numérique) pour tester si le BP est pressé (état HIGH) ou bien relâché (état LOW), cette fonction présente un seul argument avec lequel on désigne l'entrée numérique à lire.



Réalisation du montage avec résistance pull-down

En fonction de l'état du BP, la LED s'allume ou bien s'éteint, et ce avec la fonction digitalWrite qui sert à écrire un HIGH ou bien LOW sur la sortie 13.

- **Schéma du montage avec résistance pull-up :**

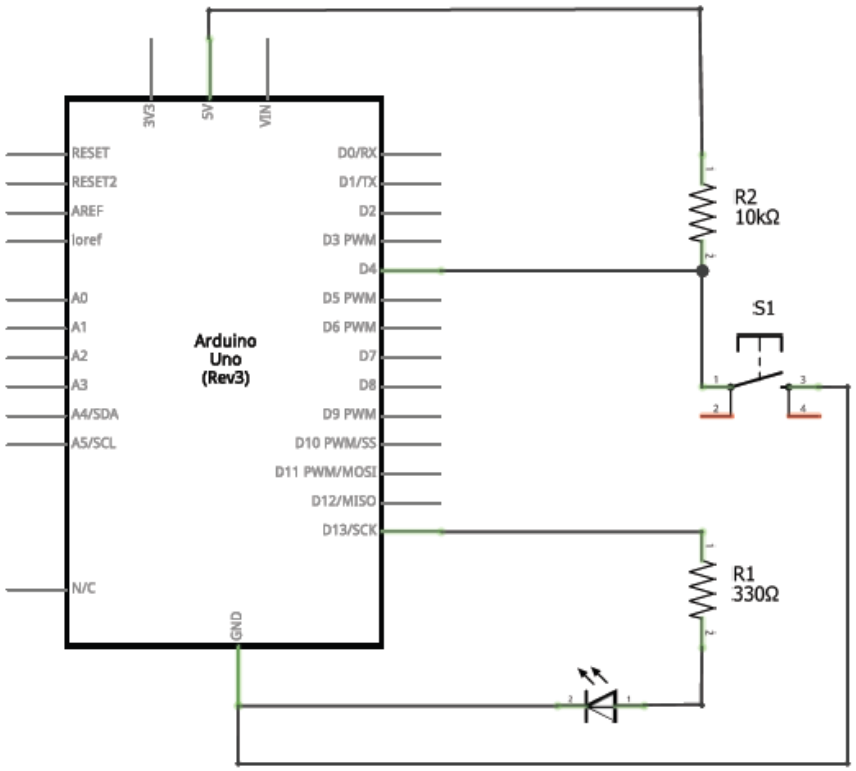
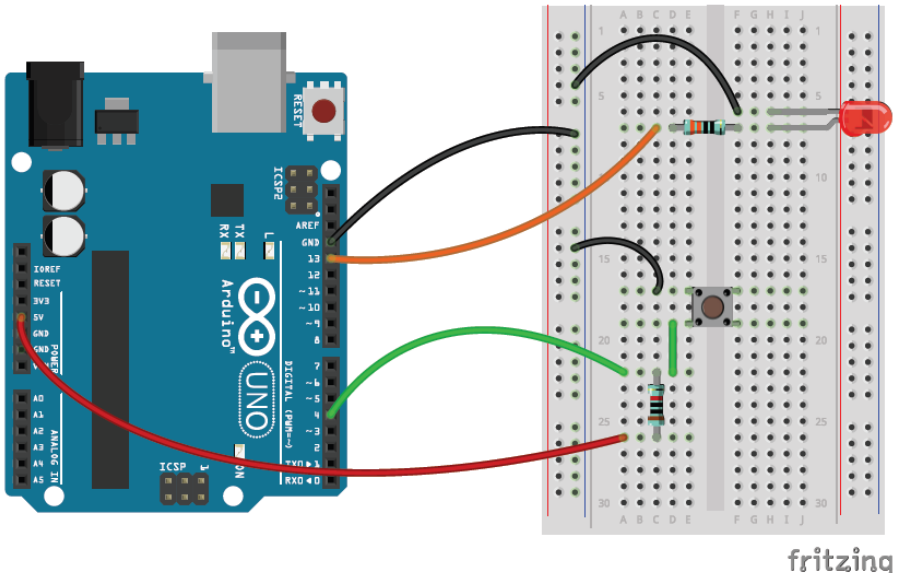


Schéma du montage

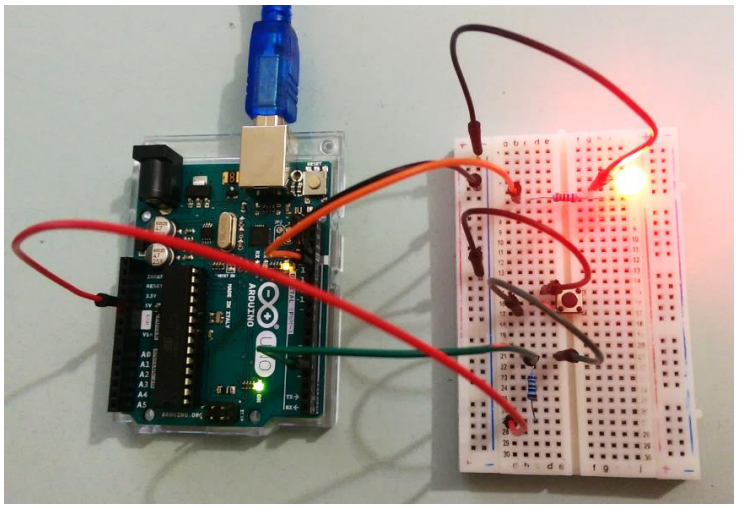


Réalisation du circuit avec fritzing

- **Code du sketch :**

Pour ce montage avec résistance pull-up, on peut garder le même sketch précédent, sauf qu'il y'aura un changement au niveau du fonctionnement : la LED commence par être allumée, et quand on appuie sur le BP elle va s'éteindre, si vous voulez obtenir le même fonctionnement du montage avec résistance pull-down il faudrait effectuer une simple modification dans le sketch : dans la fonction loop au lieu de tester si l'entrée du BP est à l'état HIGH, on la teste s'ellle est à l'état LOW, et pour ce faire, il suffit de changer la ligne « if (digitalRead(BrocheBouton) == HIGH) » par « if (digitalRead(BrocheBouton) == LOW) ».

Remarque : pour les deux montages 'pull-up' et 'pull-down', Si vous déconnectez la broche d'entrée du BP de tout, la LED peut clignoter de manière erratique. C'est parce que l'entrée est "flottante" - c'est-à-dire qu'elle retournera au hasard soit HAUT soit BAS. C'est pourquoi vous avez besoin d'une résistance pull-up ou pull-down dans le circuit.



Réalisation du montage avec résistance pull-up

- **Schéma du montage avec résistance pull-up interne :**

Les montages avec des résistances de tirage à l'état haut sont extrêmement classiques. Les fabricants de microcontrôleurs intègrent donc, en général, une résistance de tirage à l'état haut activable par logiciel sur chaque broche.

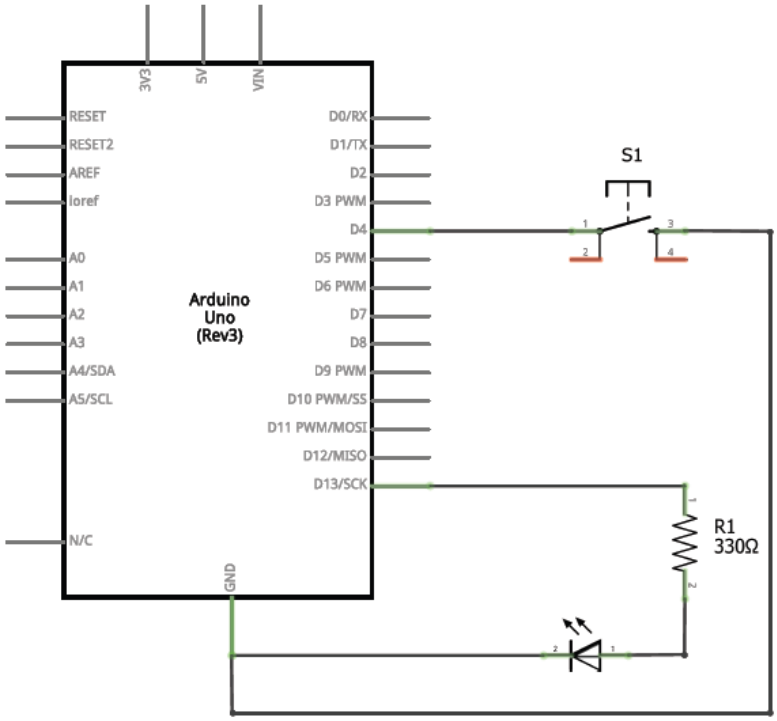
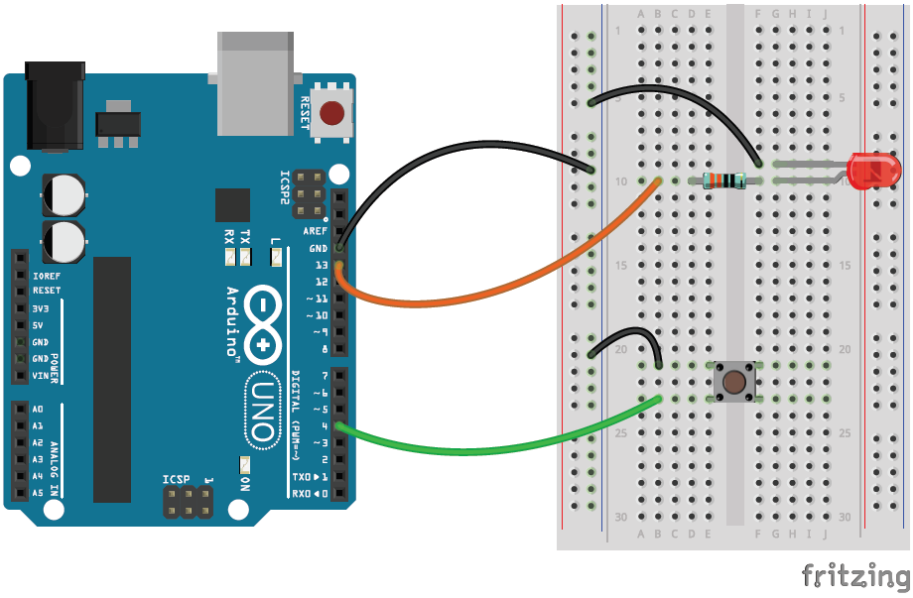


Schéma du montage

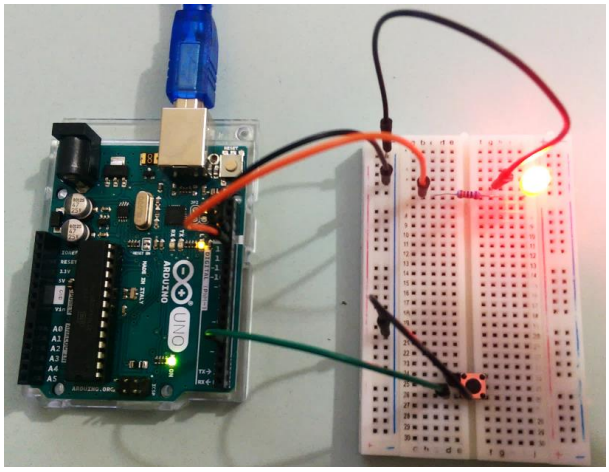


Réalisation du circuit avec fritzing

- **Code du sketch :**

```
int BrocheLed = 13;                 //Déclaration et initialisation de la broche digitale 13
int BrocheBouton = 4;               //Déclaration et initialisation de la broche digitale 4
void setup() {
pinMode(BrocheLed, OUTPUT);     // Programmation de la broche 13 comme sortie
pinMode(BrocheBouton, INPUT_PULLUP);  // Programmation de la broche 4 comme entrée et
//activation de la résistance pull-up interne
}
void loop() {
if (digitalRead(BrocheBouton) ==  HIGH) // Si l'entrée du BP est à l'état haut (BP pressé)
digitalWrite(BrocheLed, HIGH);      // Mettre la broche 13 sur le niveau haut (5V)
else                                // Sinon (l'entrée du BP est à l'état haut (BP relâché))
digitalWrite(BrocheLed, LOW);     // Mettre la broche 13 sur le niveau bas (0V)
}
```

Comme vous pouvez le constater, le code du sketch reste toujours le même que les autres précédents, à bien noter que pour la configuration de l'entrée du BP, le mode INPUT_PULLUP de la fonction pinMode(), nous permet d'activer la résistance de tirage interne. Celle-ci est d'environ 50K et ne tire donc pas beaucoup de puissance. C'est très utile car cela permet d'économiser une partie (la résistance) et un peu de câblage. Cependant, sur de longues distances de câble, le pull-up peut être trop faible pour empêcher le bruit d'être capté sur le câble.



Réalisation du montage avec résistance pull-up interne

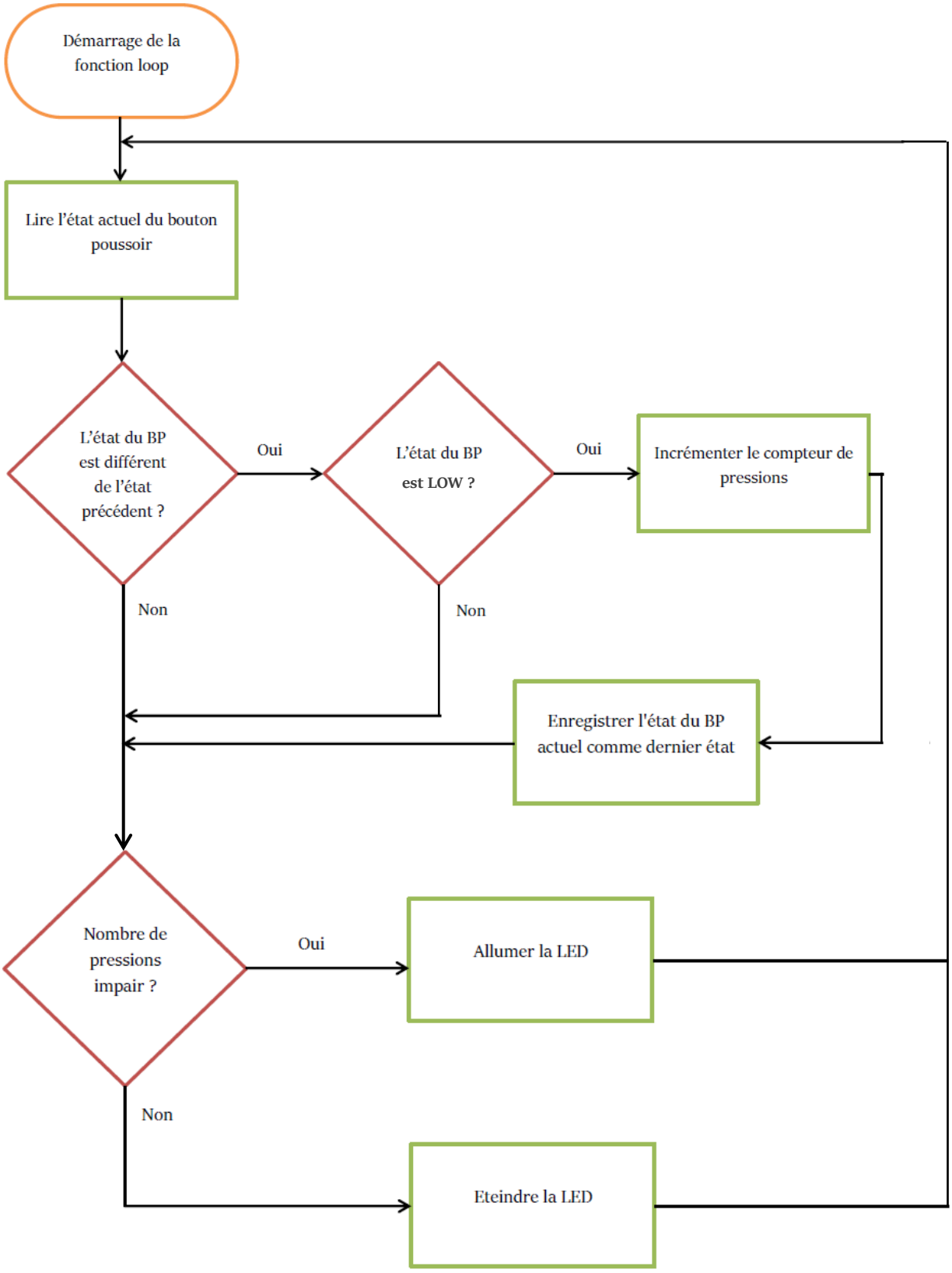
Maintenant, en se basant sur ce denier montage avec résistance pull-up interne, on veut obtenir un fonctionnement marche-arrêt comme le suivant :

- Quand on appuie sur le BP, la LED s'allume et reste allumée même si on relâche le BP.
- Quand on appuie une deuxième fois sur le BP, la LED doit s'éteindre et ainsi de suite.

- **Code du sketch pour un fonctionnement en marche-arrêt :**

```
int BrocheLed = 13;                  // Déclaration et initialisation de la broche digitale 13
int BrocheBouton = 4;                // Déclaration et initialisation de la broche digitale 4
int CompteurBouton = 0;              // Compteur pour le nombre de pressions sur le BP
int EtatBouton = 0;                  // L'état actuel du BP
int DernierEtatBouton = 0;           // L'état précédent du BP
void setup() {
pinMode(BrocheLed, OUTPUT);     // Programmation de la broche 13 comme sortie
pinMode(BrocheBouton, INPUT_PULLUP);  // Programmation de la broche 4 comme entrée et
//activation de la résistance pull-up interne
}
void loop() {
EtatBouton = digitalRead(BrocheBouton);  // Lire l'état actuel du BP
if (EtatBouton != DernierEtatBouton) {  // Comparer l'état actuel avec l'état précédent du BP
if (EtatBouton == LOW) // Si l'état a changé (a passé de HIGH à LOW)
CompteurBouton++ ; // Incrémenter le compteur
}
DernierEtatBouton = EtatBouton;    //Enregistrer l'état actuel comme dernier état, pour la
//prochaine fois à travers la fonction loop
if (CompteurBouton % 2 != 0)         // Si le reste de la division entière est différent de 0
digitalWrite(BrocheLed, HIGH);       // Mettre la broche 13 sur le niveau haut (5V)
else
digitalWrite(BrocheLed, LOW);      // Mettre la broche 13 sur le niveau bas (0V)
 }
```

- **Explications et analyse du sketch :**



Organigramme fonctionnel de la fonction loop du sketch

Pour le nouveau fonctionnement, nous avons ajoutés trois variables :

- **EtatBouton** : sert à contenir l'état actuel du bouton poussoir.
- **DernierEtatBouton** : sert à mémoriser l'état précédent du bouton poussoir.
- **CompteurBouton** : sert à stocker le nombre de pressions sur le BP.

L'organigramme de la page précédente résume le fonctionnement du sketch, tout d'abord on lit l'état actuel du bouton poussoir (HIGH ou LOW) et on le compare avec l'état précédent, si les deux sont différents, on vérifie si l'état actuel est HIGH (c'est-à-dire qu'il a passé de HIGH à LOW), si oui on incrémente le compteur, ce dernier va servir d'allumer ou bien d'éteindre la LED, en suite on enregistre l'état actuel comme état précédent pour le prochain test dans la fonction loop. L'étape suivante consiste à vérifier si le compteur du nombre de pressions sur le BP est un nombre impair, si oui la LED sera allumé et sinon la LED sera éteinte et ainsi de suite. Le cycle de fonctionnement peut être simulé comme suite :

Etat initial	pas de pressions encore	compteur : 0	Etat de la LED : LOW (éteinte)
Cycle 1	Première pression	compteur : 1	Etat de la LED : HIGH (allumée)
Cycle 2	Deuxième pression	compteur : 2	Etat de la LED : LOW (éteinte)
Cycle 3	troisième pression	compteur : 3	Etat de la LED : HIGH (allumée)
.	.	.	.
.	.	.	.
.	.	.	.

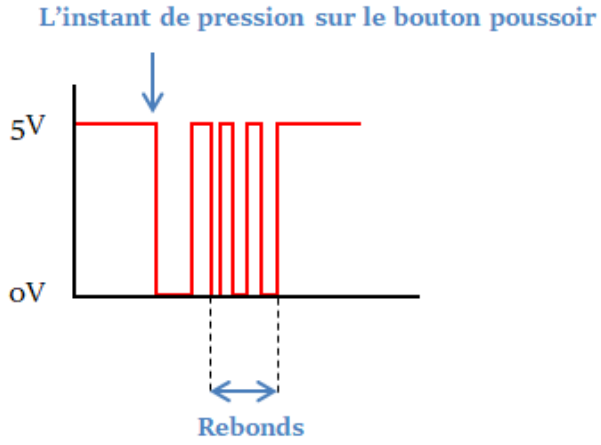
Un nombre entier est impair si et seulement si le reste de sa division entière par 2 est différent de 0. Dans le sketch nous avons utilisé l'opérateur de la fonction modulo '%' permettant de renvoyer le reste de la division entière de deux nombres entiers.

- **Les rebonds du bouton poussoir :**

Le code de notre montage fonctionne, mais vous pouvez constater de temps en temps, que notre LED s'allume ou bien s'éteint deux fois ou plus, même si le BP est appuyé une seule

fois. Cela n'est pas dû à la LED, mais bien au bouton poussoir et notamment au phénomène appelé 'rebond'.

Les boutons poussoirs sont des composants mécaniques constitués de lames en métal, ils ont des ressorts pour faire un bon contact, et ces contacts rebondissent à mesure qu'ils se ferment. Cela ne dure que quelques millisecondes et est imperceptible à l'œil nu, mais pour l'électronique cela est parfaitement visible. Le graphique suivant montre l'exemple d'un bouton poussoir appuyé :



Rebonds pour une pression sur un bouton poussoir

Au lieu d'une transition en douceur de 5V (via la résistance pull-up) à oV(ground), nous voyons une série de rebonds. Si nous avions juste du code simple, nous pourrions penser que quelqu'un a appuyé sur le BP 4 fois.

Ce serait très difficile à utiliser, si vous appuyez une fois sur le BP pour allumer la LED, et encore une fois pour l'éteindre, si le nombre de rebonds est pair (nombre de pressions pair), alors la LED restera éteinte.

Il existe deux façons de résoudre le problème des rebonds.

Solution logicielle :

Une technique simple consiste simplement d'attendre un court délai (par exemple entre 10 et 50 millisecondes), après la lecture de l'état du BP afin d'éviter les rebonds.

Le code du nouveau sketch y compris le délai anti-rebond est le suivant :

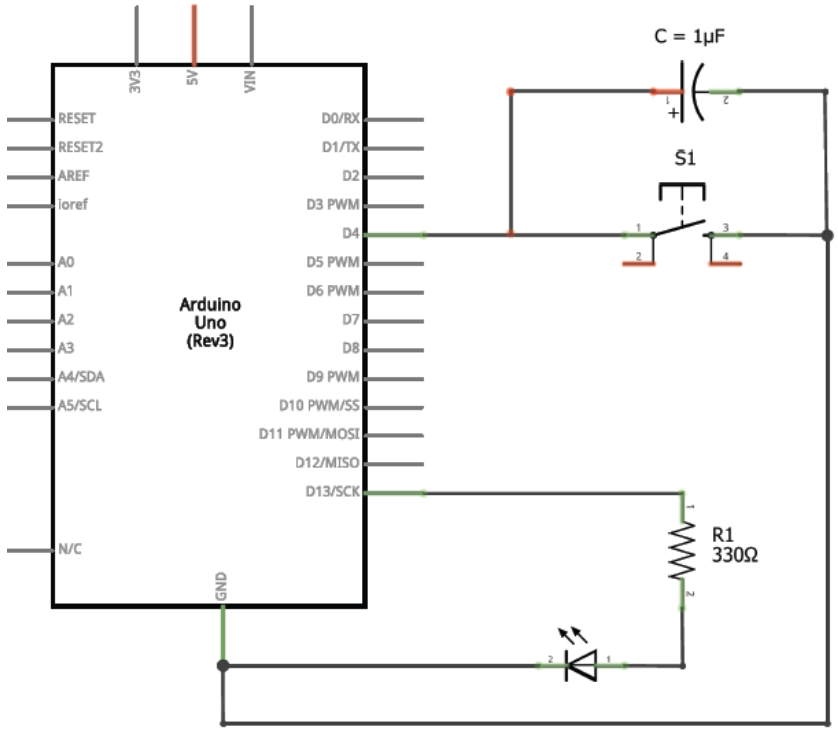
```
int BrocheLed = 13;                 // Déclaration et initialisation de la broche digitale 13
int BrocheBouton = 4;               // Déclaration et initialisation de la broche digitale 4
int CompteurBouton = 0;             // Compteur pour le nombre de pressions sur le BP
int EtatBouton = 0;                 // L'état actuel du BP
int DernierEtatBouton = 0;          //L'état précédent du bouton
void setup() {
pinMode(BrocheLed, OUTPUT);     // Programmation de la broche 13 comme sortie
pinMode(BrocheBouton, INPUT_PULLUP);  // Programmation de la broche 4 comme entrée et
//activation de la résistance pull-up interne
}
void loop() {
EtatBouton = digitalRead(BrocheBouton);  // Lire l'état actuel du BP
if (EtatBouton != DernierEtatBouton) {  // Comparer l'état actuel avec l'état précédent du //BP
if (EtatBouton == LOW) // Si l'état a changé (a passé de LOW à HIGH)
CompteurBouton++ ; // Incrémenter le compteur
delay(20) ;                 //Attendre 20 ms pour éviter les rebonds
}
DernierEtatBouton = EtatBouton;    //Enregistrer l'état actuel comme dernier état, pour la
//prochaine fois à travers la fonction loop
if (CompteurBouton % 2 != 0)          // Si le reste de la division entière est différent de 0
digitalWrite(BrocheLed, HIGH);       // Mettre la broche 13 sur le niveau haut (5V)
else
digitalWrite(BrocheLed, LOW);       // Mettre la broche 13 sur le niveau bas (0V)
 }
```

Avec un délai de 20 ms qui parait suffisant dans notre cas, la période des rebonds sera dépassée, et le fonctionnement souhaitable sera obtenu.

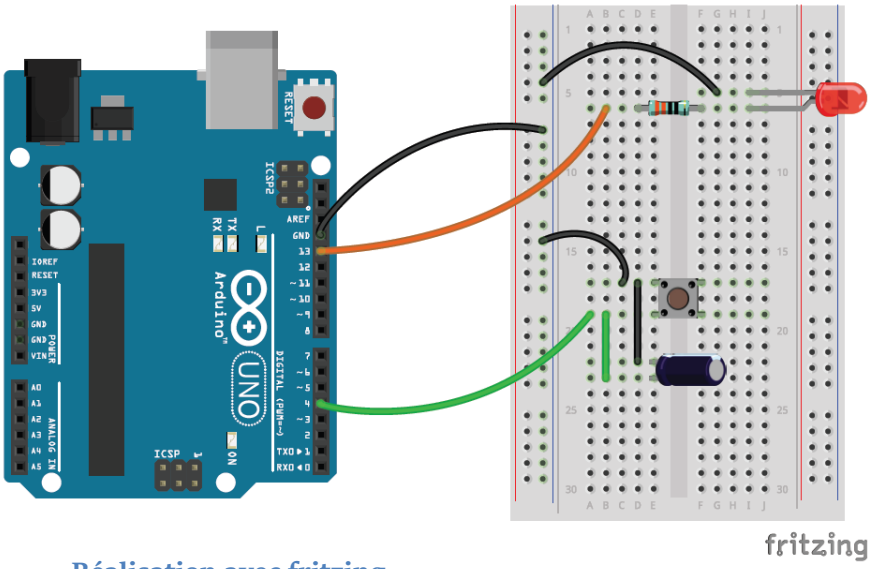
Il y en n'a pas que cette solution unique bien sûr, vous pouvez trouver d'autres dans la littérature. Pourtant il existe des bibliothèques spécifiques au rebond qui sont prêtes à l'utilisation.

Solution matérielle :

Une deuxième technique consiste à brancher un condensateur en parallèle avec le bouton poussoir (figure ci-dessous) :



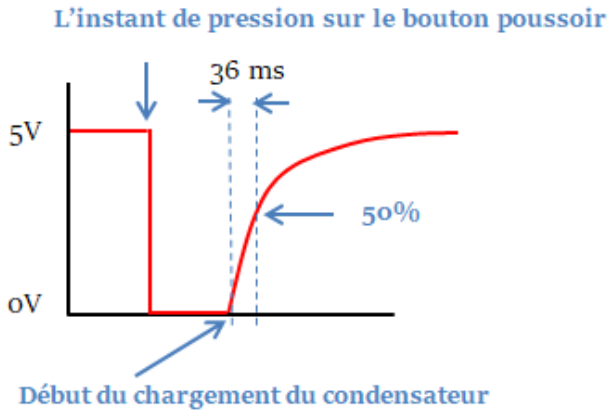
Branchement d'un condensateur pour récompenser le rebond



Réalisation avec fritzing

Dans ce cas, le condensateur se charge via la résistance de pull-up interne. Lorsque l'interrupteur est enfoncé, il se décharge et prend un moment pour se recharger. Ce retard détourne effectivement les rebonds, de sorte que le premier code (sans délai anti-rebond) pourrait être utilisé.

Le graphique suivant montre le chargement du condensateur une fois que le BP est relâché.



Correction des rebonds pour une pression sur un bouton

Etant donné que 50% environ est considéré comme état HIGH, il nous a donné environ 36 ms de retard anti-rebond. Ainsi, un condensateur plus petit pourrait être utilisé si le délai anti-rebond souhaité est plus court.

Pourquoi faut-il 36 ms pour charger le condensateur à 50%?

Le condensateur se charge à 50% en `0.69 * R * C` où R est (environ) 50kΩ, C = 1 µF (0.000001 F), donc le temps est:

```
0.69 * 50000 * 0.000001 = 0.0345 seconds (34.5 ms)
```

Étant donné que le chiffre de 50k pour la résistance de pull-up interne est une estimation, cela semble à peu près correct.

Pour ce faire, nous regardons la formule pour la constante RC :

```
V(t) = V(o) * (1 - e^(-t / RC) )
```

En supposant que V (o), la tension initiale est égale à 1 pour simplification, nous avons:

```
V = 1 - e^(-t / RC)
1 - V = e^(-t / RC)
ln (1 - V) = -t / RC
t = - RC * ln(1 - V)
```

Où "ln" est le logarithme népérien (log sur la base e). On peut donc en déduire que le temps d'atteindre la moitié de la tension finale sera :

```
R = 50000  (resistance 50k)
C = 0.000001  (condensateur 1 µF)
V = 0.5    (moitié de la tension initiale)
t = - R * C  * ln (1 - V)
t = - 50000 * 0.000001 * ln (1 - 0.5)
t = 0.03465 seconds (34.6 ms)
```

RC est par ailleurs connu sous le nom "constante du temps" (τ) qui est le temps nécessaire pour charger un condensateur, à travers une résistance, à environ 63.2% de sa valeur initiale.

Formule générale du retard d'un circuit RC :

La formule générale du retard est :

```
t = - ln ( (V - Vc) / V) * R * C
```

Pour calculer la tension après un temps spécifique (t), utilisez cette formule:

```
Vc = V - (V * exp (-t / (R * C)))
Avec :
V = tension d'alimentation (tension initiale)
Vc = tension de sortie (tension cible)
```

R = résistance en ohms

C = capacité en farads

t = temps en secondes

Constante du temps RC :

On s'est mis d'accord que : $\tau = RC$

Un condensateur se chargera (ou se déchargera) à 63.2% de τ .

Le chiffre de 63.2% provient de :

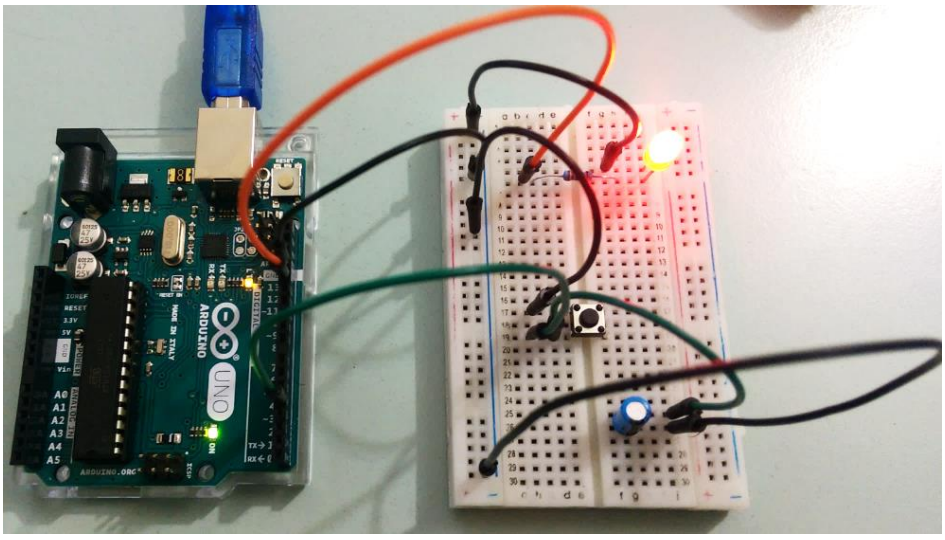
1 – (exp^-1) qui est d'environ 0.63212055882856

Un condensateur est considéré comme ayant complètement chargé (ou complètement déchargé selon votre sens de travail) dans 5τ, donc vous pouvez donc calculer que pour une résistance et un condensateur donné, il faudra:

5 * R * C

En fait, il se charge / décharge à 99.3% de ce temps, dont 99.3% provient de :

1 – (exp^-5) qui est d'environ 0.99326205300091



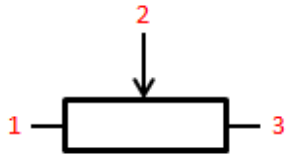
Réalisation du montage avec branchement d'un condensateur 1 μF

L'utilisation d'un potentiomètre pour varier l'intensité de lumière :

Ce petit montage, va montrer l'utilité d'une sortie PWM. Il permet de faire changer l'intensité de lumière d'une LED en fonction d'une tension appliquée à une entrée analogique de la carte Arduino.

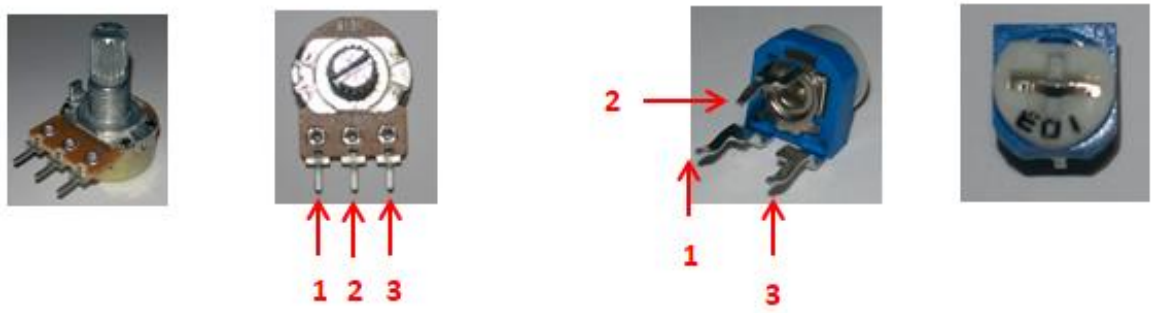
- **Le potentiomètre :**

Un potentiomètre est un type de résistance variable à trois bornes, dont une est reliée à un curseur se déplaçant sur une piste résistante terminée par les deux autres bornes.



Symbole d'un potentiomètre

Ce système permet de recueillir, entre la borne reliée au curseur et une des deux autres bornes, une tension qui dépend de la position du curseur et de la tension à laquelle est soumise la résistance.



Deux types de potentiomètres

Matériels et composants nécessaires :

Aduino Uno

Breadboard

LED

Résistance 330 Ω

Fils de connexion

Potentiomètre 10 KΩ

- **Schéma du montage :**

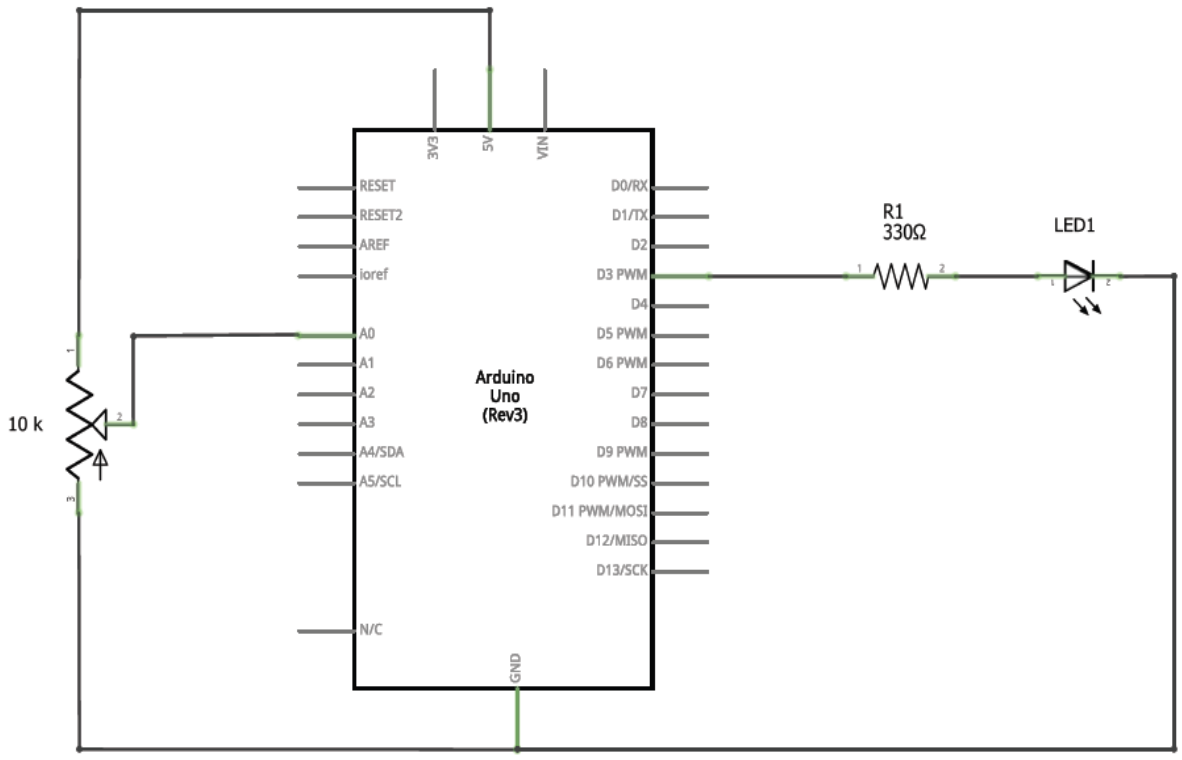
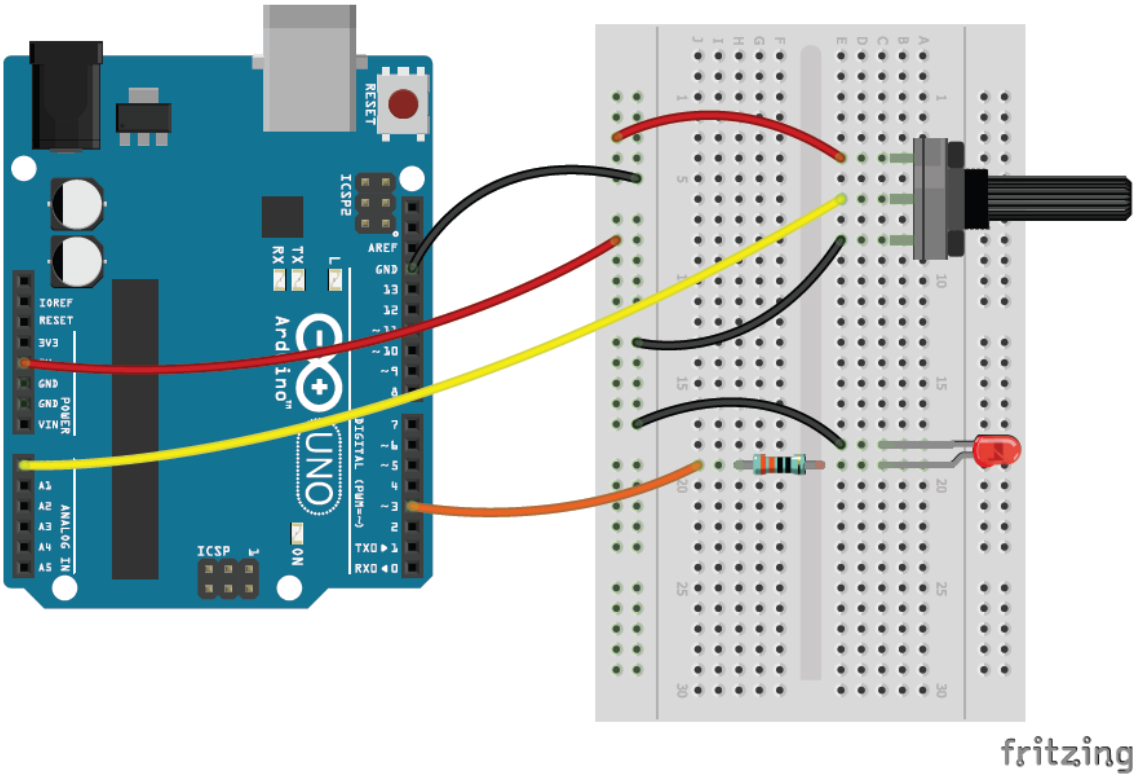


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

```
int BrocheLed = 3;     // LED connectée à la sortie numérique 3 (PWM)
int EntreeAnalog = 0;   // Entrée analogique 0 liée au potentiomètre
int val = 0;        // Variable pour stocker la valeur de l'entrée analogique
void setup()
{
pinMode(BrocheLed, OUTPUT);   // Programmation de la broche 3 comme sortie
}
void loop()
{
 val = analogRead(EntreeAnalog);   // Lire la valeur à l'entrée analogique 0
 analogWrite(BrocheLed, val / 4);  // Valeurs de analogRead peuvent aller de 0 à 1023, et
//celles de analogWrite  de 0 à 255
}
```

- **Explications et analyse du sketch :**

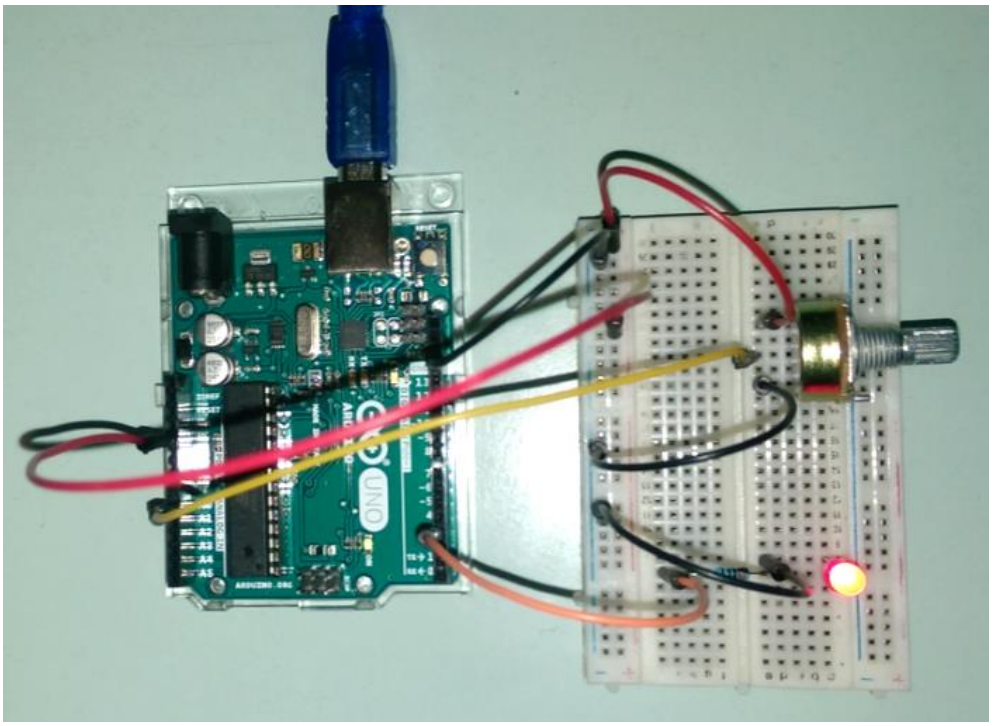
Le potentiomètre lié à l'entrée analogique 0 de la carte délivre une tension analogique de 0 à 5V, Cette tension est convertit par le CAN (**C**onvertisseur **A**nalogique **N**umérique) de la carte, et donne un nombre de conversion compris entre 0 et 1023. L'instruction analogRead permet de lire ce nombre.

On rappelle que La carte Arduino contient un convertisseur analogique-numérique 10 bits à 6 canaux (8 canaux sur le Mini et Nano, 16 sur le Mega). Cela signifie qu'il va mapper les tensions d'entrée entre 0 et 5 volts en valeurs entières entre 0 et 1023. Cela donne une résolution entre des lectures de : 5 volts / 1024 unités ou 0.049 volts (4.9 mV) par unité. La plage d'entrée et la résolution peuvent être modifiées en utilisant analogReference ().

Il faut environ 100 microsecondes (0.0001 s) pour lire une entrée analogique, de sorte que le taux de lecture maximum est d'environ 10 000 fois par seconde.

Ensuite, Pour la sortie analogique 3, au moyen d'un signal PWM (MLI : **M**odulation de la **L**argeur d'**I**mpulsion), on a utilisé l'instruction analogWrite, qui nécessite de spécifier le

numéro de la broche et la valeur à convertir en signal PWM. Cette valeur peut varier entre 0 et 255 (max), la raison pour laquelle on a divisé le nombre de conversion sur 4 (**1024/4 = 255)**.



Réalisation du montage

Clignotement d'une LED :

Pour faire clignoter une LED pendant une durée précise, nous allons travailler sur le même premier montage (HELLO WORLD) et effectuer des modifications sur le code du sketch.

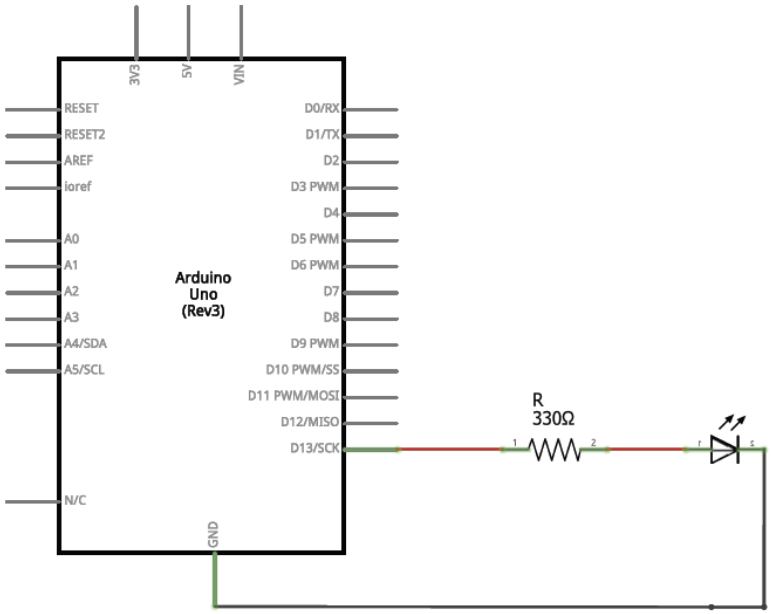


Schéma du montage

- **Code du sketch :**

```
int BrocheLed = 13;                 //Déclaration et initialisation de la broche digitale 13
unsigned  int Tempo = 1000;     //Déclaration et initialisation d'une temporisation de 1000 ms
void setup() {
pinMode(BrocheLed, OUTPUT);  // Programmation de la broche 13 comme sortie
}
void loop() {
digitalWrite(BrocheLed, HIGH);   // Mettre la broche 13 sur le niveau haut (5V)
delay(Tempo);                          // Déclencher la temporisation
digitalWrite(BrocheLed, LOW);   // Mettre la broche 13 sur le niveau bas (0V)
delay(Tempo);                          // Déclencher la temporisation
}
```

Avec ce nouveau code, vous obtiendrez une LED clignotante pendant une durée de 1000 millisecondes.

- **Explications et analyse du sketch :**

Dans ce nouveau code, nous avons ajouté une variable 'tempo' de type entier, initialisée à une valeur de 1000. Dans la fonction loop, nous avons appelé deux fois la fonction **delay,** celle-ci sert à la temporisation, elle interrompt l'exécution du sketch pour un temps correspondant à la valeur donnée qui exprime la durée en millisecondes (ms). 1000 signifie une attente de 1000 ms soit 1 seconde avant de continuer. Voici les différentes étapes du fonctionnement du sketch :

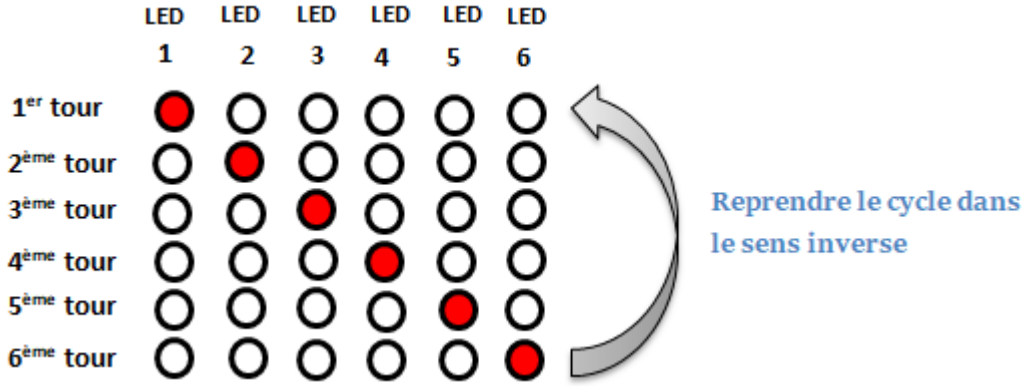
1. Allumer la LED liée à la broche 13.
2. Attendre 1 seconde.
3. Eteindre la LED liée à la broche 13.
4. Attendre 1 seconde.
5. Revenir à l'étape 1 pour recommencer.

Commande d'un séquenceur de lumière à deux vitesses :

Vous maitrisez maintenant suffisamment les LED pour être capable de faire des montages dans lesquels clignotent plusieurs LED. Nous allons apprendre, dans un premier temps à réaliser un séquenceur de lumière qui fait allumer l'une après l'autre, les différentes LED.

Nous essayerons de chercher par la suite à optimiser le fonctionnement du montage en y introduisant des boutons poussoirs.

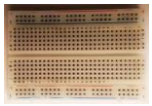
Dans notre montage les LED devront s'allumer conformément au modèle suivant :



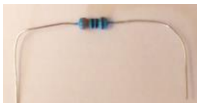
Modèle de fonctionnement du séquenceur de lumière

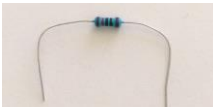
Matériels et composants nécessaires :

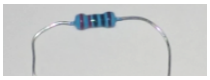

Aduino Uno


Breadboard


6 * LED


6 * Résistance 330 Ω


2 * Résistance 10 kΩ


2 * Résistance 220 Ω


Fils de connexion


2 * Bouton poussoir NO


2 * Condensateur 1 µF

- **Schéma du montage :**

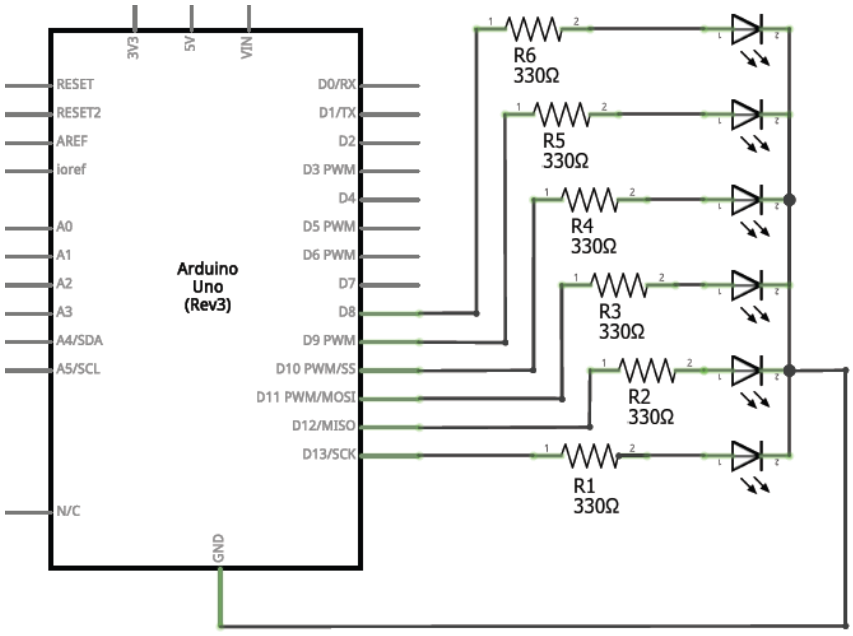
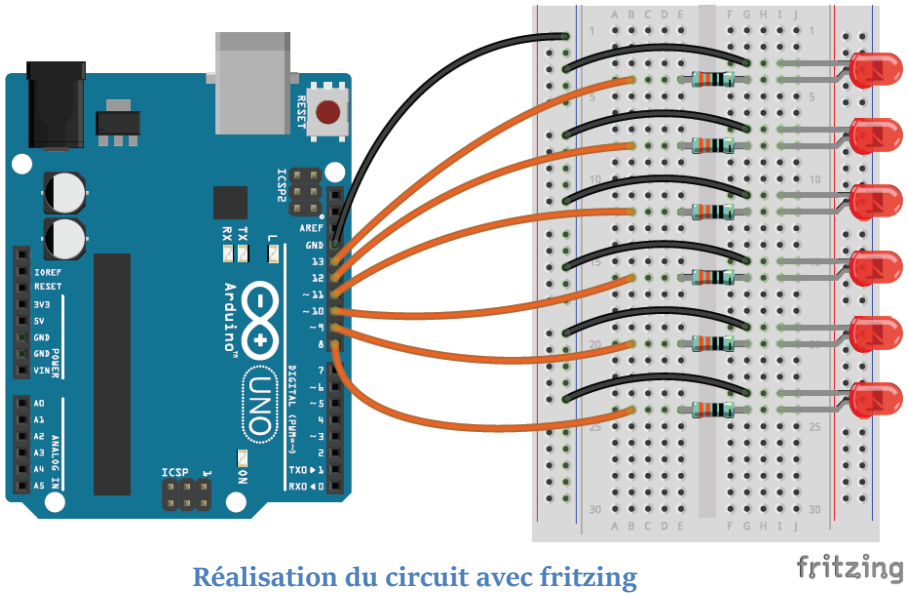


Schéma du montage



Réalisation du circuit avec fritzing

- **Code du sketch :**

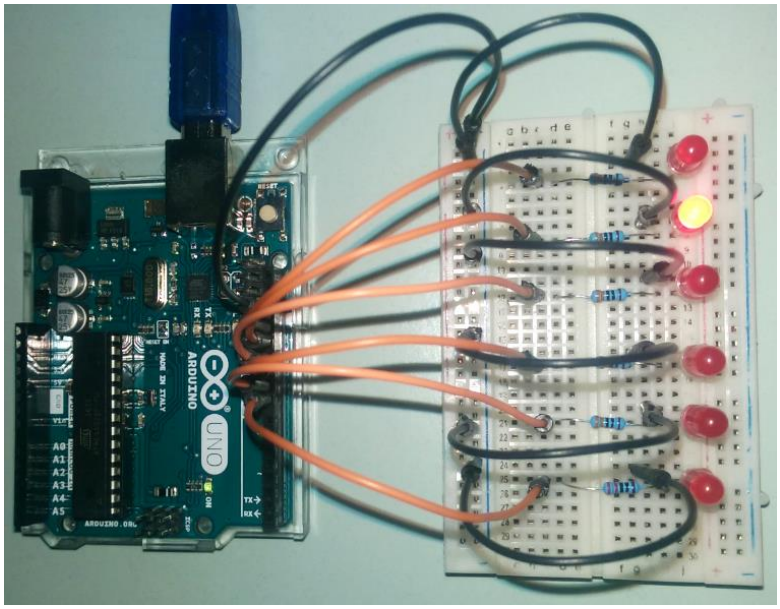
```
int BrocheLed[] = {8, 9, 10, 11, 12, 13};  // Tableau de LED avec numéros des broches
unsigned int Tempo = 500; // Pause entre les changements en ms
void setup(){
for (int i = 0; i < 6; i++)
 pinMode(BrocheLed[i], OUTPUT); // Programmation des broches du tableau comme sorties
}
void loop(){
for(int i = 0; i < 6; i++){                  //Boucle du sens direct
digitalWrite(BrocheLed[i], HIGH); // Élément du tableau au niveau HIGH
delay(Tempo);
digitalWrite(BrocheLed[i], LOW);  // Élément du tableau au niveau LOW
 }
 for(int i = 4; i > 0; i--){                   //Boucle du sens inverse
 digitalWrite(BrocheLed[i], HIGH); // Élément du tableau au niveau HIGH
 delay(Tempo);
 digitalWrite(BrocheLed[i], LOW);  // Élément du tableau au niveau LOW
 }
}
```

- **Explications et analyse du sketch :**

Dans le sketch de ce montage nous avons servi la notion des tableaux.

En première ligne, nous avons initialisé un tableau de LED avec les différentes broches à utiliser. Et, afin de les programmer toutes en sorties, une boucle for est utilisée pour parcourir les éléments du tableau.

La deuxième boucle sert à commander les LED pour un cycle du séquenceur dans le sens direct. Ainsi, la troisième boucle se charge de la commande du cycle dans le sens inverse, vous remarquez bien que celle-ci commence avec le $5^{ème}$ élément (i=4) correspondant à la LED numéro 5 au lieu de commencer avec le $6^{ème}$ élément. Cette technique est optée afin de garder la même durée d'allumage de chaque LED pour un cycle entier (un cycle dans le sens direct + un cycle dans le sens inverse).



Réalisation du séquenceur

- **Optimisation du fonctionnement avec des boutons poussoirs :**

Maintenant nous allons effectuer des améliorations sur le plan matériel et logiciel de manière à obtenir le fonctionnement suivant :

- Le séquenceur démarre ou bien s'arrête en fonction de la pression sur un premier bouton poussoir de marche-arrêt.
- Le séquenceur change de vitesse, autrement dit il change la durée d'allumage des LED en fonction de la pression sur un deuxième bouton poussoir, on obtiendra donc deux vitesse de fonctionnement à l'aide du BP (v1-v2).

- **Schéma du montage :**

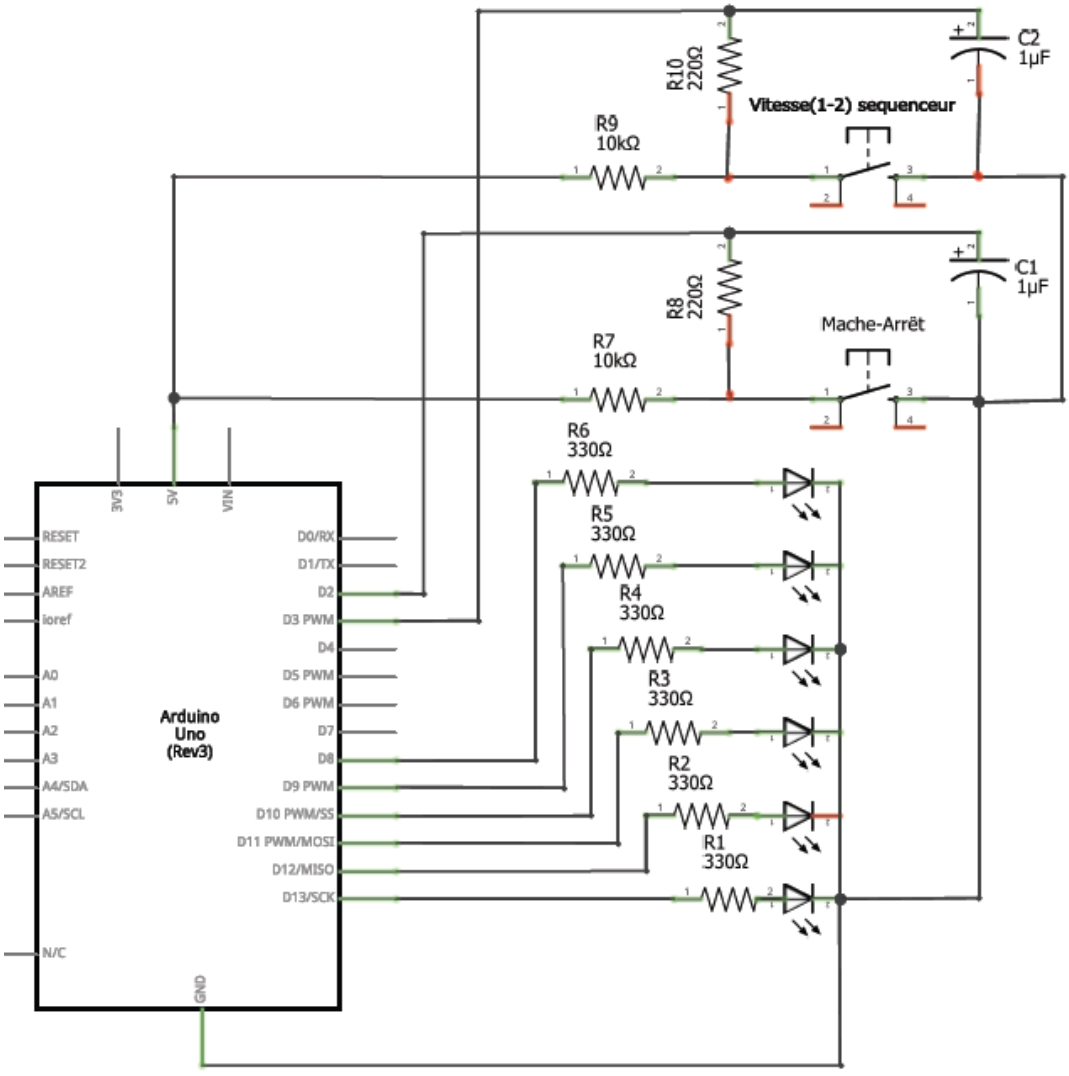
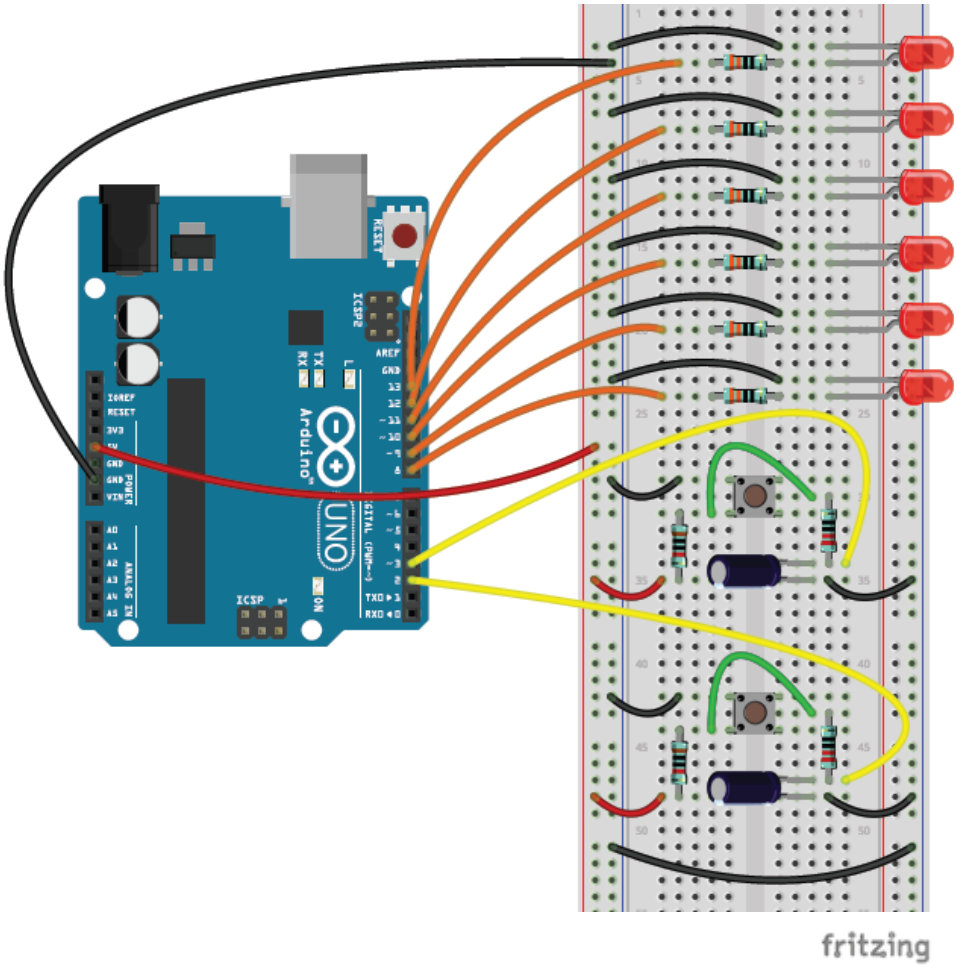
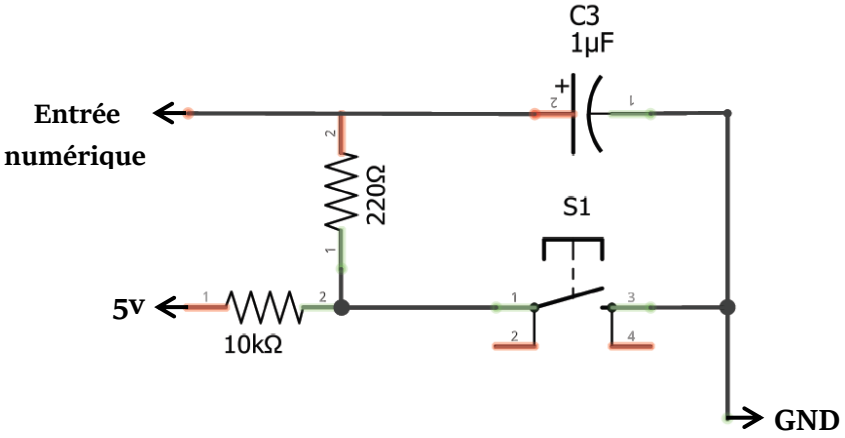


Schéma du montage



Réalisation du montage avec fritzing

Le circuit de commande est basé sur deux boutons poussoirs, chaqu'un est branché sur un montage avec résistance pull-up, y inclut un condensateur anti-rebonds. (Figure ci-dessous).



Montage du BP avec résistance pull-up et condensateur anti-rebonds

Lorsqu'on alimente ce circuit, le condensateur commence à se charger via la résistance (10kΩ + 220Ω), et lorsqu'on appuie sur le bouton poussoir il se décharge via la résistance 220Ω.

- **Code du sketch :**

```
int BrocheLed[] = {8, 9, 10, 11, 12, 13};  // Tableau de LED avec numéros des broches
unsigned int Tempo = 500;    // Pause entre les changements en ms
unsigned int Tempo1 = 500; //  Première vitesse
unsigned int Tempo2 = 100;  // Deuxième vitesse
int Bp1 = 2;                    // Entrée du bouton poussoir 1
int Bp2 = 3;                    // Entrée du bouton poussoir 2
volatile int  k = LOW;          //Indice pour mémoriser les pressions sur le BP1
volatile int  b = LOW;          //Indice pour mémoriser les pressions sur le BP2
int i;                          // Indice courant pour les boucles de fonctionnement
void setup(){
for (int i = 0; i < 6; i++)
pinMode(BrocheLed[i], OUTPUT); // Programmation des broches du tableau comme sorties
attachInterrupt (digitalPinToInterrupt (Bp1), MarcheArret, FALLING); // Attachement de
//l'entrée numérique  BP1 au porogramme d'interruption 'MarcheArret' (démarrage sur front
//descendant)
attachInterrupt (digitalPinToInterrupt (Bp2), Vitesse, FALLING); // Attachement de l'entrée
//numérique  BP2 au porogramme d'interruption 'Vitesse' (démarrage sur front  descendant)
}
void loop(){
for(int i = 0; i < 6 && k == HIGH; i++){              //Boucle du sens direct
digitalWrite(BrocheLed[i], HIGH); // Élément du tableau au niveau HIGH
delay(Tempo);
digitalWrite(BrocheLed[i], LOW);  // Élément du tableau au niveau LOW
 }
 for(int i = 4; i > 0 && k == HIGH; i--){              //Boucle du sens inverse
```

```
digitalWrite(BrocheLed[i], HIGH); // Élément du tableau au niveau HIGH
delay(Tempo);
digitalWrite(BrocheLed[i], LOW);  // Élément du tableau au niveau LOW
 }
}
 void MarcheArret() {    //Programme d'interruption traitant le démarrage et l'arrêt du
// séquenceur
 k = !k;                        // Basculement de l'indice k  de l'état actuel à l'état opposé
 if (k == LOW)
  digitalWrite(BrocheLed[i], LOW);    // Eteindre le séquenceur
 }
void Vitesse(){          //Programme d'interruption traitant les deux vitesses du séquenceur
b = !b;                        // Basculement de l'indice b de l'état actuel à l'état opposé
if (b == LOW)
Tempo = Tempo1;     // Première vitesse
else
Tempo = Tempo2;    // Deuxième vitesse
 }
```

- **Explications et analyse du sketch :**

L'amélioration du sketch régisse principalement dans l'intégration des deux programme appelés d'interruptions afin de démarrer le séquenceur ou bien de changer sa vitesse de fonctionnement.

Programme d'interruption :

Les programmes de service d'interruptions ou ISR (**I**nterrupt **S**ervice **R**outine) nous permettent de répondre aux événements "externes" tout en faisant autre chose.

Les ISR sont des types spéciaux de fonctions qui ont des limitations uniques que la plupart des autres fonctions n'ont pas. Un ISR ne peut avoir aucun paramètre, et ne devrait rien retourner.

Généralement, un ISR devrait être aussi court et rapide que possible. Si votre sketch utilise plusieurs ISR, un seul peut s'exécuter à la fois, d'autres interruptions seront exécutées après la fin de l'actuelle dans un ordre qui dépend de la priorité dont elles disposent. La fonction millis() repose sur les interruptions pour compter, donc il ne s'incrémentera jamais dans un ISR. En outre la fonction delay() nécessite des interruptions pour fonctionner, elle ne fonctionnera pas s'elle est appelée dans un ISR, micros() fonctionne initialement, mais commencera à se comporter de façon erratique après 1-2 ms. delayMicroseconds() n'utilise aucun compteur, donc il fonctionnera normalement.

Les variables globales sont utilisées pour transmettre des données entre un ISR et le programme principal. Pour assurer que les variables partagées entre un ISR et le programme principal sont mises à jour correctement, ils doivent être déclarés comme volatiles.

Syntaxe :

attachInterrupt (digitalPinToInterrupt (broche), ISR, mode); (recommandé)

attachInterrupt (interruption, ISR, mode); (non recommandé)

attachInterrupt (broche, ISR, mode); (non recommandé, utilisée pour Arduino Due, Zero, MKR1000, et 101 uniquement)

Paramètres :

Interruption : le numéro de l'interruption (int).

Broche : le numéro de broche (Arduino Due, Zero, MKR1000 uniquement).

ISR : l'ISR à appeler lorsque l'interruption se produit; cette fonction ne doit prendre aucun paramètre et ne retourne rien. Cette fonction est parfois appelée routine de service d'interruption.

Mode : définit quand l'interruption doit être déclenchée. Quatre constantes sont prédéfinies en tant que valeurs valides :

'LOW' pour déclencher l'interruption quand la broche est à l'état bas.

'CHANGE' pour déclencher l'interruption chaque fois que la broche change d'état.

'RISING' pour se déclencher quand la broche passe de l'état bas à l'état haut.

'FALLING' lorsque la broche passe de l'état haut à l'état bas.

Les cartes Due, Zero et MKR1000 permettent également : '**HIGH'** pour déclencher l'interruption lorsque la broche est à l'état haut.

N.B : La carte Uno dispose de deux broches numériques qui peuvent être programmés comme entrées d'interruption : broche 2 et 3.

Les ISR utilisés dans le sketch :

Pour notre sketch nous avons créé deux ISR, le premier appelé « MarcheArret » , sert à démarrer ou bien d'arrêter le séquenceur lorsque le BP1 est pressé. Le deuxième ISR permet de changer la vitesse du séquenceur à chaque pression sur le BP2.

Dans la fonction setup(), l'instruction « attachInterrupt(digitalPinToInterrupt(Bp1), MarcheArret, FALLING); » nous permet d'attacher notre ISR « MarcheArret » à la broche numérique 2 (l'entrée liée au bouton poussoir BP1), et de définir le mode « FALLING » de déclenchement de l'interruption, qui est correspondant au circuit pull-up du BP1.

De la même façon, l'instruction « attachInterrupt(digitalPinToInterrupt (Bp2), Vitesse, FALLING); » attache l'ISR « Vitesse » à la broche 3 avec un mode « FALLING » de déclenchement de l'interruption.

Définition de l'ISR « MarcheArret » :

```
void MarcheArret() {
 k = !k;
 if (k == LOW)
  digitalWrite(BrocheLed[i], LOW);
 }
```

Nous avons servi la variable 'k', déclarée comme volatile int en début du sketch et initialisé à l'état LOW (arrêt) pour mémoriser chaque pression sur le BP1.

A l'état initial (k=LOW), la condition sur les boucle de fonctionnement (k==HIGH) est non valide, la boucle for ne peut démarrer et le séquenceur est donc en arrêt.

Lorsqu'on appuie une seule fois sur le BP1, la broche 2 passe de l'état haut à l'état bas, notre ISR « MarcheArret » s'exécute. La variable 'k' passe de son état actuel (LOW) à l'état opposé (HIGH), la condition sur les boucles for est maintenant valide, le séquenceur démarre.

Lorsqu'on appuie une deuxième fois sur le BP1, la variable 'k' change son état actuel (HIGH) et passe à l'état LOW, la LED allumée d'indice i dans la boucle 'for' en cours sera immédiatement éteinte, le séquenceur s'arrête.

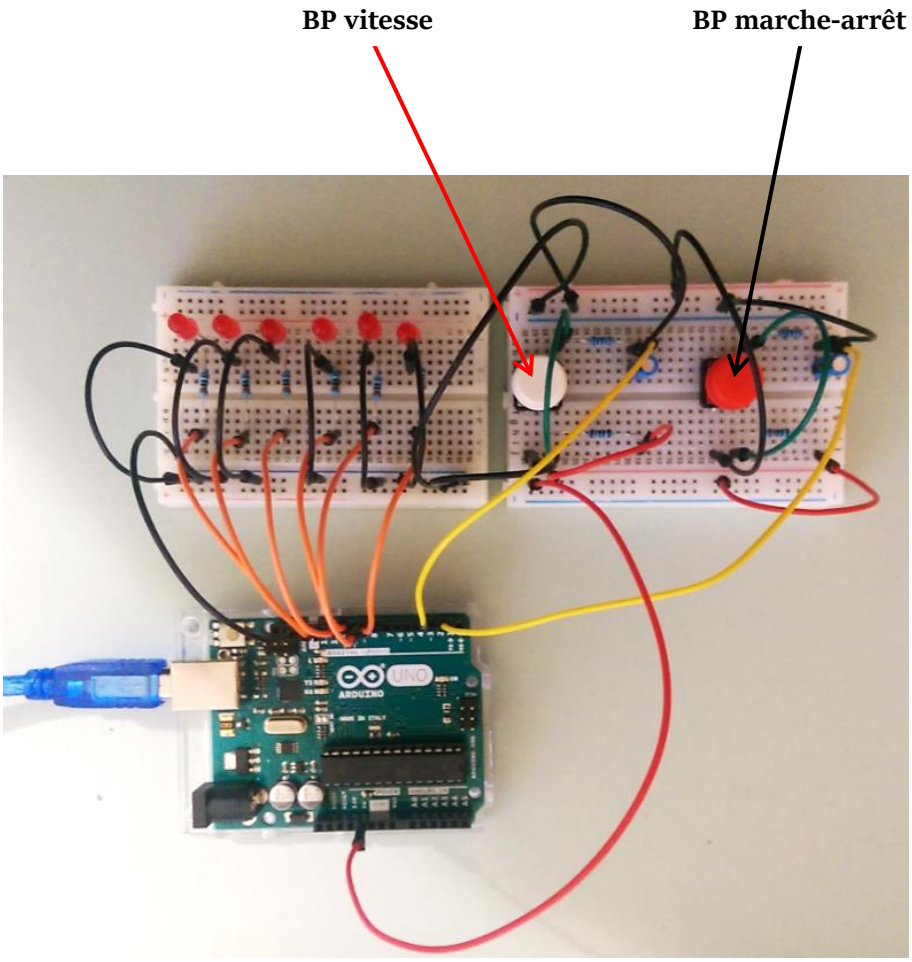
Définition de l'ISR « Vitesse » :

```
void Vitesse(){
b = !b;
if (b == LOW)
Tempo = Tempo1;
else
Tempo = Tempo2;
 }
```

La variable 'b', quant à elle, permet de mémoriser les pressions sur le BP2.

Lorsque le BP2 est appuyé une seule fois la broche numérique 3 passe de l'état HIGH à l'état LOW, l'ISR « Vitesse » s'exécute. La variable 'b' passe à l'état HIGH et la variable 'Tempo' équivalente au délai d'allumage de chaque LED change sa valeur initiale (500) et devient 'Tempo2' (100), le séquenceur fonctionne plus vite.

Si le BP2 est appuyé une deuxième fois, 'b' passe à l'état opposé (LOW), la variable 'Tempo' devient 'Tempo1'(500), le séquenceur ralentit.



Réalisation de la commande du séquenceur avec deux boutons poussoirs

Application sur les LED RGB:

Les LED RGB sont largement utilisées dans de nombreux projets avec la plateforme Arduino, Ils travaillent sur le concept de base de la combinaison des couleurs de base : rouge, vert et bleu. Toutes les couleurs ont ces composants de couleur élémentaires en eux. Ainsi, nous pouvons faire n'importe quelle couleur que nous voulons en utilisant une LED RGB.

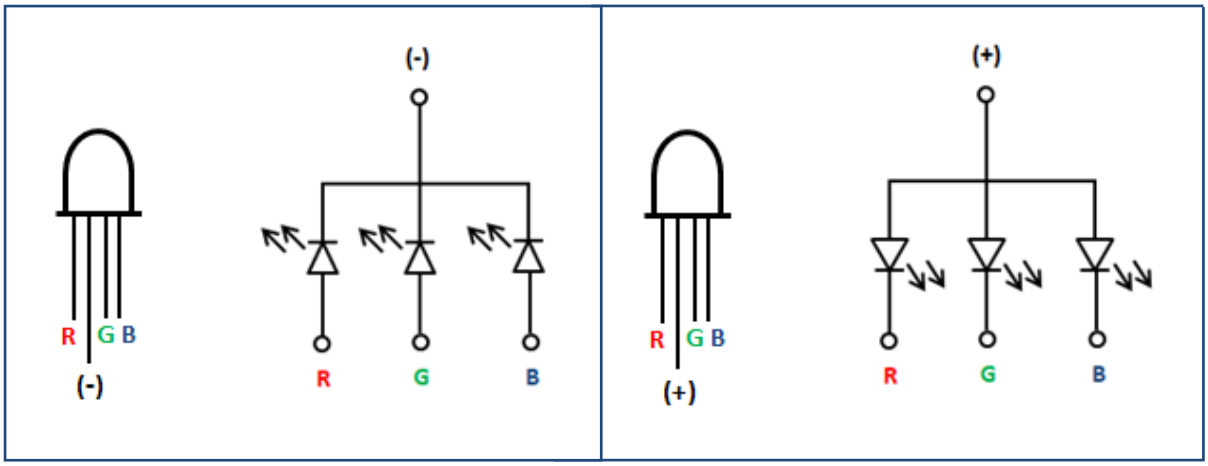
Dans cette partie nous allons réaliser en premier temps un montage permettant de varier la couleur de luminosité d'une LED RGB. On appliquera ensuite, le montage du séquenceur

précédent sur les LED RGB, afin d'obtenir un séquenceur de lumière multi-couleurs avec une apparence spéciale.

- **Qu'est-ce qu'une LED RGB ? :**

Une LED RGB se compose de trois petites LED de trois couleurs primaires (rouge, vert et bleu) où une borne est commune pour les trois. Certaines ont une borne positive commune (anode) et d'autres ont une borne négative commune (cathode). Quand une tension différente est appliquée sur différentes LED, elles font un mélange et peuvent produire des milliers de couleurs.

LED RGB



LED RGB à cathode commune

LED RGB à anode commune

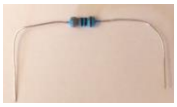
- **Commande d'une LED RGB avec trois potentiomètres**

Matériels et composants nécessaires :

Aduino Uno

Breadboard

LED RGB



3 * Résistance 330 Ω

3 * potentiomètre 10 kΩ

Fils de connexion

- **Schéma du montage :**

Dans notre cas, nous allons travailler avec des LED RGB à cathode commune. Comme j'ai dit précédemment, une LED RGB se compose de 3 petites LED. Chaqu'une de ces LED sera commandée par un potentiomètre pour varier son intensité de lumière, de manière à obtenir une combinaison de couleur différente à chaque variation d'un potentiomètre.

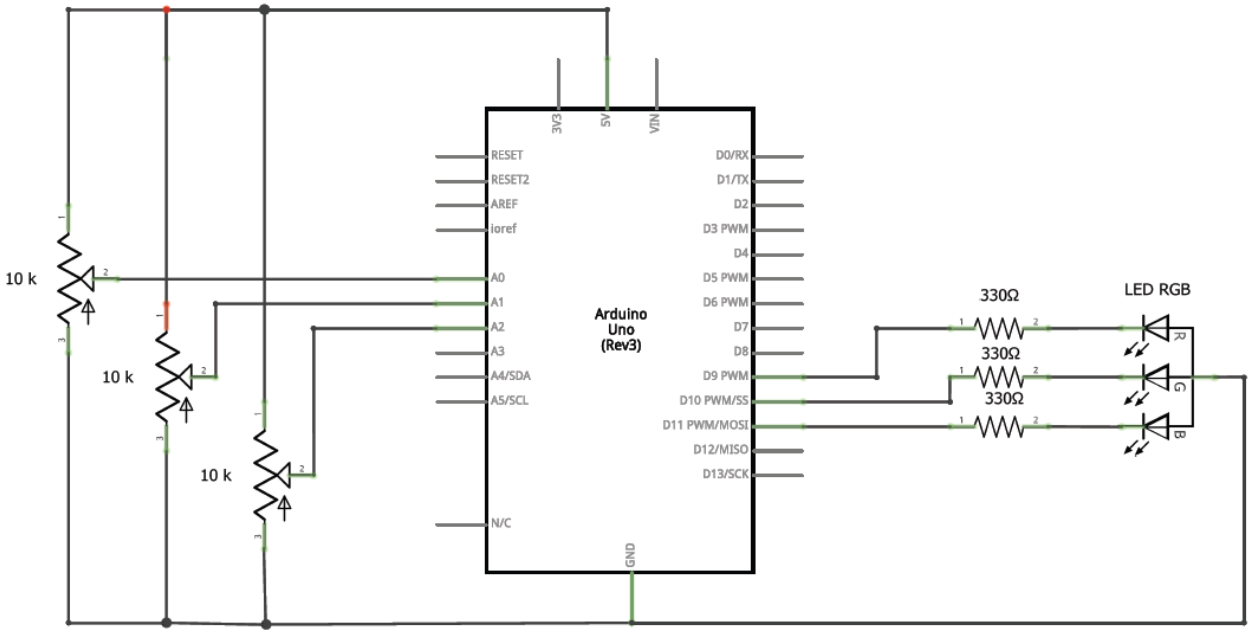
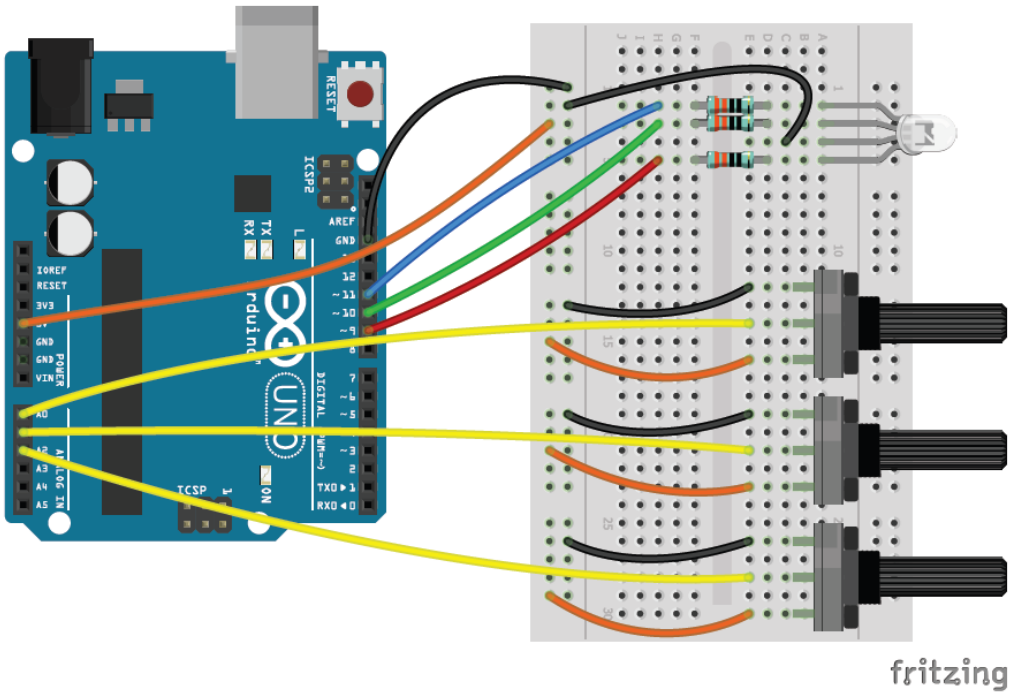


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

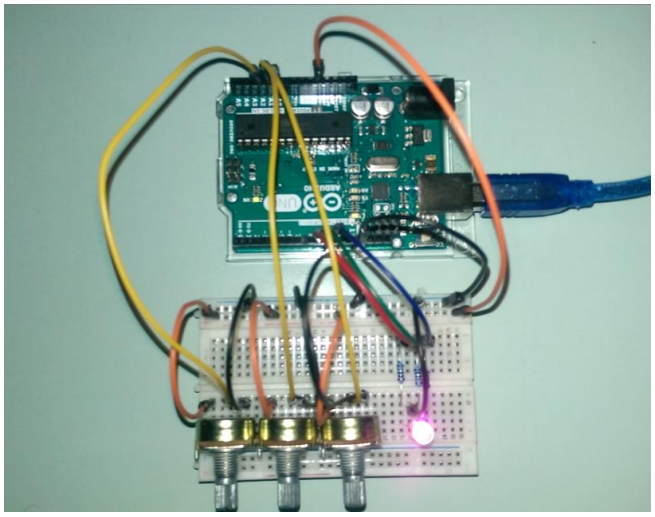
```
int BrocheLed1 = 9;     // Petite LED connectée à la sortie numérique 9 (PWM)
int BrocheLed2 = 10;    // Petite LED connectée à la sortie numérique 10 (PWM)
int BrocheLed3 = 11;    // Petite LED connectée à la sortie numérique 10 (PWM)

int AnalogLed1 = 0;   // Entrée analogique 0
int AnalogLed2 = 1;    // Entrée analogique 1
int AnalogLed3 = 2;   // Entrée analogique 2

int val1 = 0;        // Variable pour stocker la valeur de l'entrée analogique 0
int val2 = 0;       // Variable pour stocker la valeur de l'entrée analogique 1
int val3 = 0;       // Variable pour stocker la valeur de l'entrée analogique 2

void setup()
{
 pinMode(BrocheLed1, OUTPUT);   // Programmation de la broche 9 comme sortie
 pinMode(BrocheLed2, OUTPUT);   // Programmation de la broche 10 comme sortie
 pinMode(BrocheLed3, OUTPUT);   // Programmation de la broche 11 comme sortie
}
void loop()
{
 val1 = analogRead(AnalogLed1);   // Lire la valeur à l'entrée analogique 0
 val2 = analogRead(AnalogLed2);   // Lire la valeur à l'entrée analogique 1
 val3 = analogRead(AnalogLed3);   // Lire la valeur à l'entrée analogique 2

 analogWrite(BrocheLed1, val1 / 4);
 analogWrite(BrocheLed2, val2 / 4);
 analogWrite(BrocheLed3, val3 / 4);
}
```

- **Explication et analyse du sketch :**

Le principe de fonctionnement de ce sketch est basé sur celui qu'on a déjà vu dans la section « **Utilisation d'un potentiomètre pour une variation de l'intensité de lumière d'une LED** », sauf qu' ici on a affaire à trois LED.

Pour chaque petite LED, il s'agit de lire sur l'entrée analogique, la valeur de conversion de la tension délivrée par le potentiomètre correspondant, et d'écrire ainsi au moyen de la MLI, cette valeur divisée par 4 (allant de 0 à 255) sur la sortie PWM souhaitée.

En jouant sur les trois potentiomètres, on peut avoir des milliers de combinaisons de couleurs différentes.

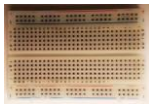


Réalisation du montage

- **Séquenceur de lumière avec des LED RGB :**

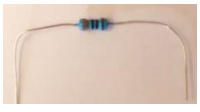
Matériels et composants nécessaires :

Aduino Uno

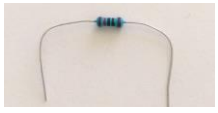


Breadboard

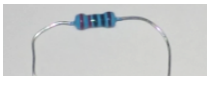
3 * LED RGB



9 * Résistance 330 Ω



2 * Résistance 10 kΩ



2 * Résistance 220 Ω

Fils de connexion

2 * Bouton poussoir NO

2 * Condensateur 1 µF

- **Schéma du montage :**

Le circuit de commande restera le même que celui du montage « séquenceur de lumière » précédent. Nous avons connecté trois LED RGB avec des sorties numériques d'Arduino, ce qui est équivalent à neuf petites LED (trois rouges, trois verts et trois bleus).

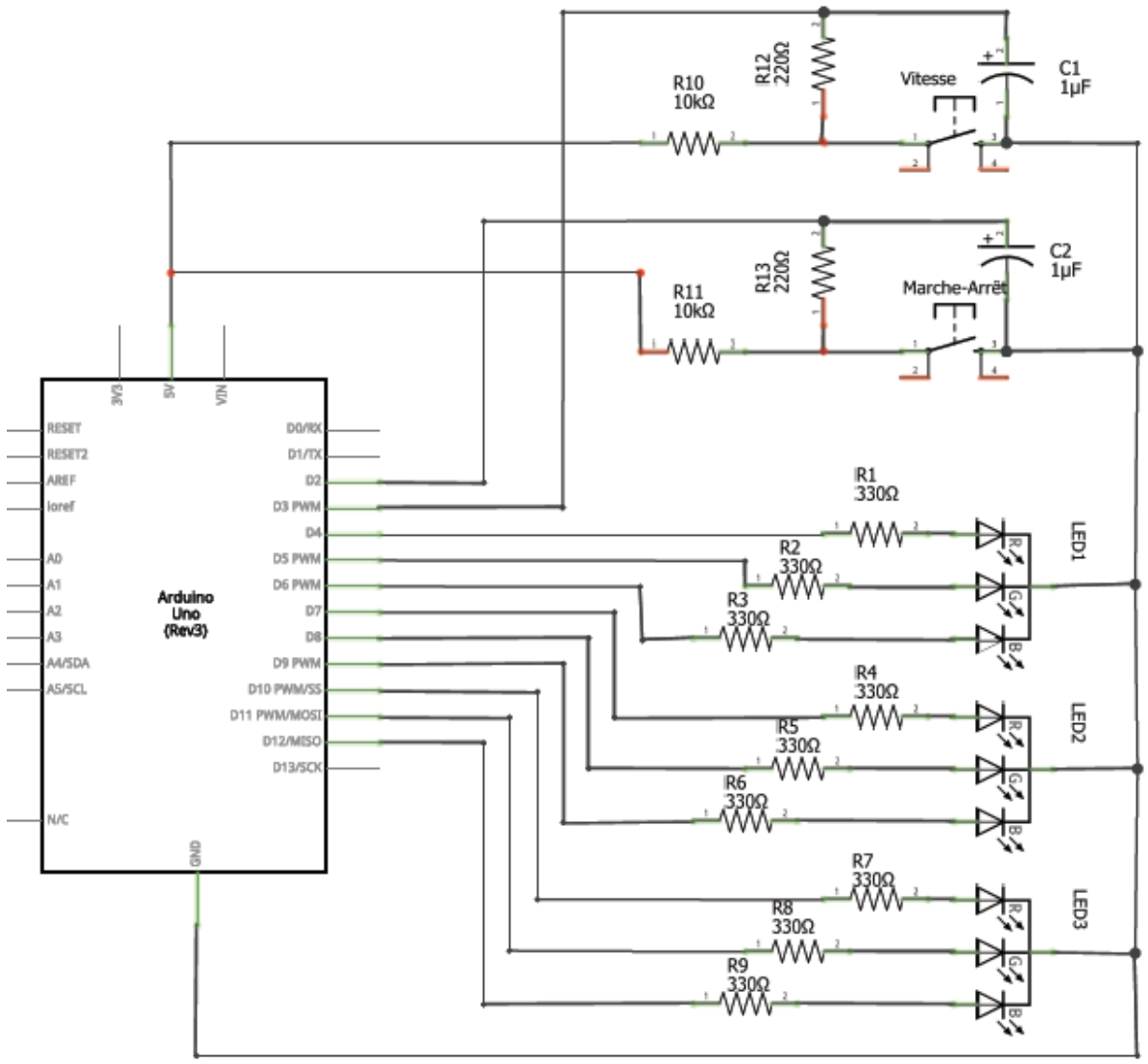
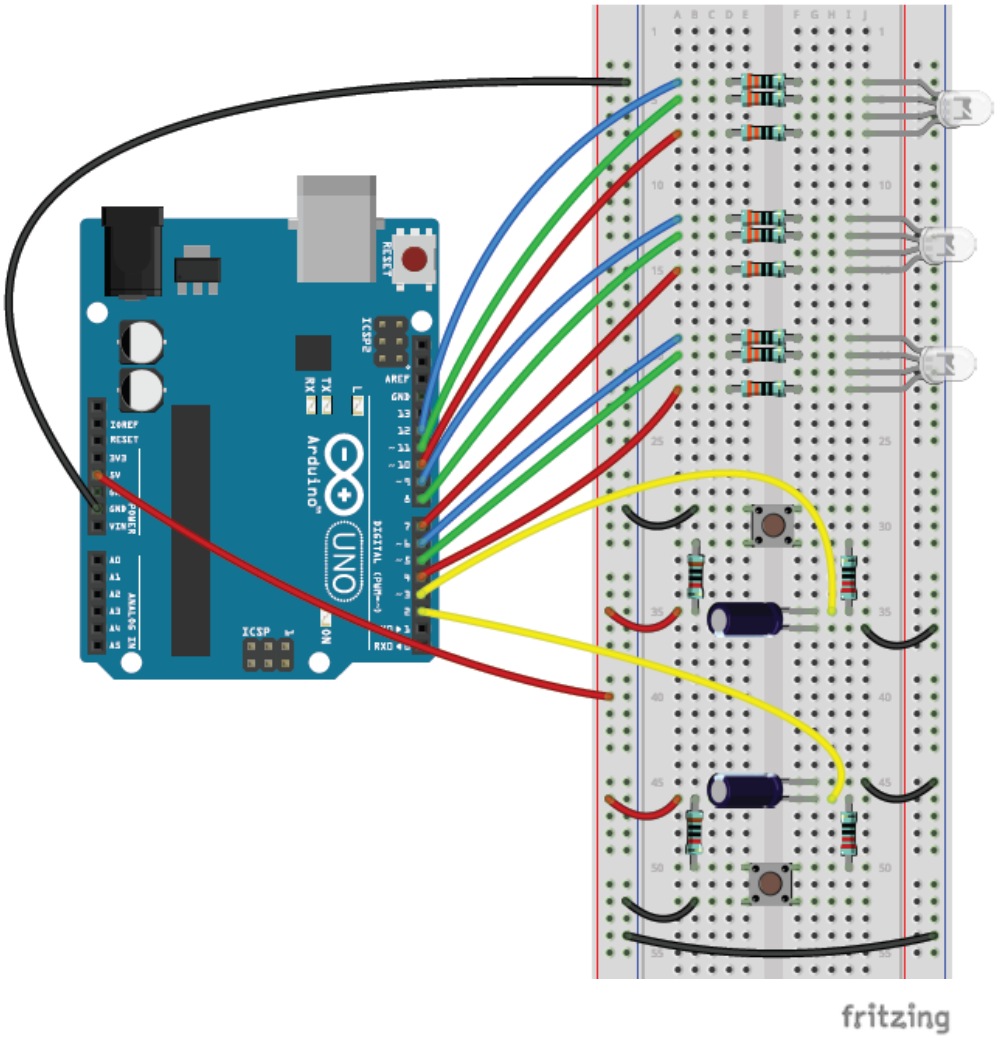


Schéma du montage



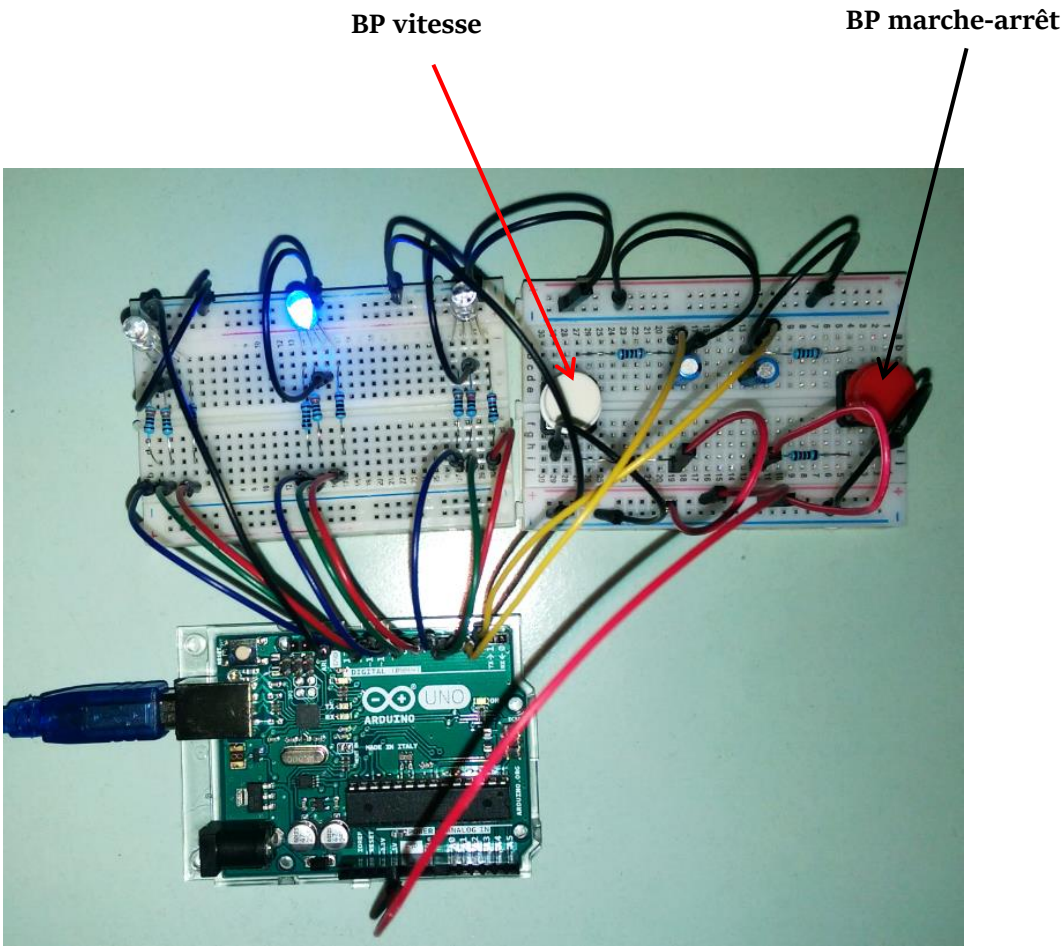
Réalisation du circuit avec fritzing

- **Code du sketch :**

```
int BrocheLed[] = {4, 5, 6, 7, 8, 9, 10, 11, 12};  // Tableau de LED avec numéros des broches
unsigned int Tempo = 500;    // Pause entre les changements en ms
unsigned int Tempo1 = 500; //  Première vitesse
unsigned int Tempo2 = 100;  // Deuxième vitesse
int Bp1 = 2;                        // Entrée du bouton poussoir 1
int Bp2 = 3;                        // Entrée du bouton poussoir 2
volatile int  k = LOW;          //Indice pour mémoriser les pressions sur le BP1
```

```
volatile int  b = LOW;          //Indice pour mémoriser les pressions sur le BP2

int i;                          // Indice courant pour les boucles de fonctionnement

void setup(){

for (int i = 0; i < 9; i++)

pinMode(BrocheLed[i], OUTPUT); // Programmation des broches du tableau comme sorties

attachInterrupt (digitalPinToInterrupt (Bp1), MarcheArret, FALLING); // Attachement de
//l'entrée numérique  BP1 au porogramme d'interruption 'MarcheArret' (démarrage sur front
//descendant)

attachInterrupt (digitalPinToInterrupt (Bp2), Vitesse, FALLING); // Attachement de l'entrée
//numérique  BP2 au porogramme d'interruption 'Vitesse' (démarrage sur front  descendant)

}

void loop(){

for(int i = 0; i < 9 && k == HIGH; i++){              //Boucle du sens direct

digitalWrite(BrocheLed[i], HIGH); // Élément du tableau au niveau HIGH

delay(Tempo);

digitalWrite(BrocheLed[i], LOW);  // Élément du tableau au niveau LOW

 }

 for(int i = 7; i > 0 && k == HIGH; i--){     //Boucle du sens inverse

 digitalWrite(BrocheLed[i], HIGH);        // Élément du tableau au niveau HIGH

 delay(Tempo);

 digitalWrite(BrocheLed[i], LOW);        // Élément du tableau au niveau LOW

 }

}

 void MarcheArret() {    //Programme d'interruption traitant le démarrage et l'arrêt du
// séquenceur

 k = !k;                        // Basculement de l'indice k  de l'état actuel à l'état opposé

 if (k == LOW)

  digitalWrite(BrocheLed[i], LOW);    // Eteindre le séquenceur

 }

void Vitesse(){          //Programme d'interruption traitant les deux vitesses du séquenceur
```

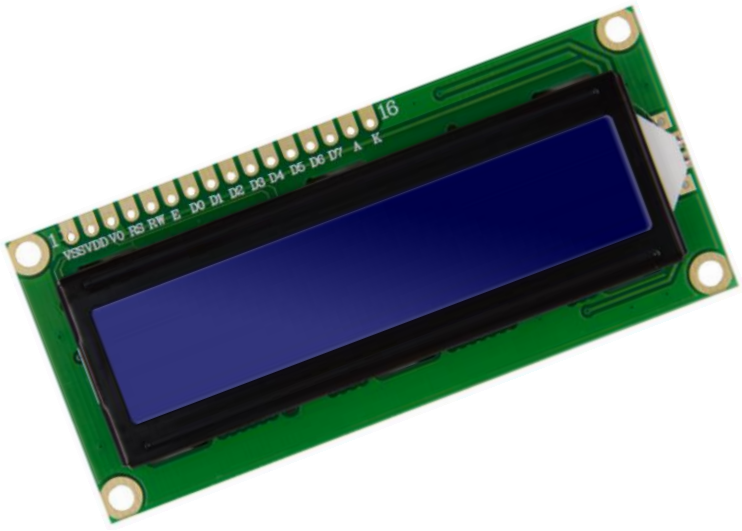
```
b = !b;                    // Basculement de l'indice b de l'état actuel à l'état opposé
if (b == LOW)
Tempo = Tempo1;     // Première vitesse
else
Tempo = Tempo2;     // Deuxième vitesse
 }
```



Réalisation du séquenceur à base de LED RGB

LEÇON 2

L’afficheur alphanumérique



Au sommaire de cette 2ème leçon :

L'afficheur LCD :

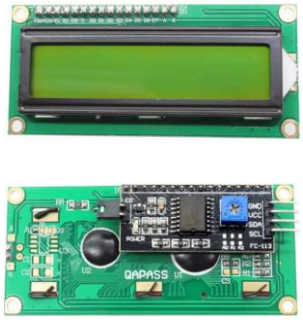
Les afficheurs LCD sont très utilisés dans les montages à microcontrôleur et permettent une grande convivialité. Dans cette leçon nous allons apprendre à mettre en œuvre un afficheur LCD 16*02 en le commandant avec la carte Uno pour afficher les caractères souhaités.

- **Qu'est-ce qu'un afficheur LCD ? :**

Un afficheur LCD (**L**iquid **C**rystal **D**isplay) ou afficheur à cristaux liquides, est un module électronique compact intelligent et nécessite peu de composants externes pour un bon fonctionnement. Il consomme relativement peu (de 1 à 5 mA).

Les LCD sont relativement bons marchés et s'utilisent avec beaucoup de facilité, ils contiennent des cristaux liquides capables de modifier leur orientation en fonction d'une tension appliquée, et de jouer plus ou moins sur l'incidence de la lumière.

Les éléments d'affichage utilisent en générale des motifs composés de points (Dot-Matrix) pour représenter à peu près tous les signes (chiffres, lettres ou caractères spéciaux).



Afficheur LCD 16*02 avec interface I2C 3 broches

Afficheur LCD Nokia 5510 84*84

Afficheur LCD 16*02 avec interface standard 16 broches

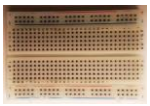
Plusieurs afficheurs sont disponibles sur le marché et diffèrent les uns des autres, non seulement par leurs dimensions, mais aussi par leurs caractéristiques techniques et leur tension de service. Certains sont dotés d'un rétroéclairage de l'affichage.

Commande d'un afficheur LCD avec un Arduino Uno :

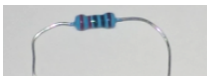
Dans ce montage nous allons travailler avec un afficheur LCD à 2 lignes de 16 caractères chacune (16*02), avec interface standard de 16 broches.

Matériels et composants nécessaires :

Aduino Uno



Breadboard



Résistance 220 Ω

Afficheur LCD 16*02 avec interface standard 16 broches

Potentiomètre 10 KΩ

Fils de connexion

- **Schéma du montage :**

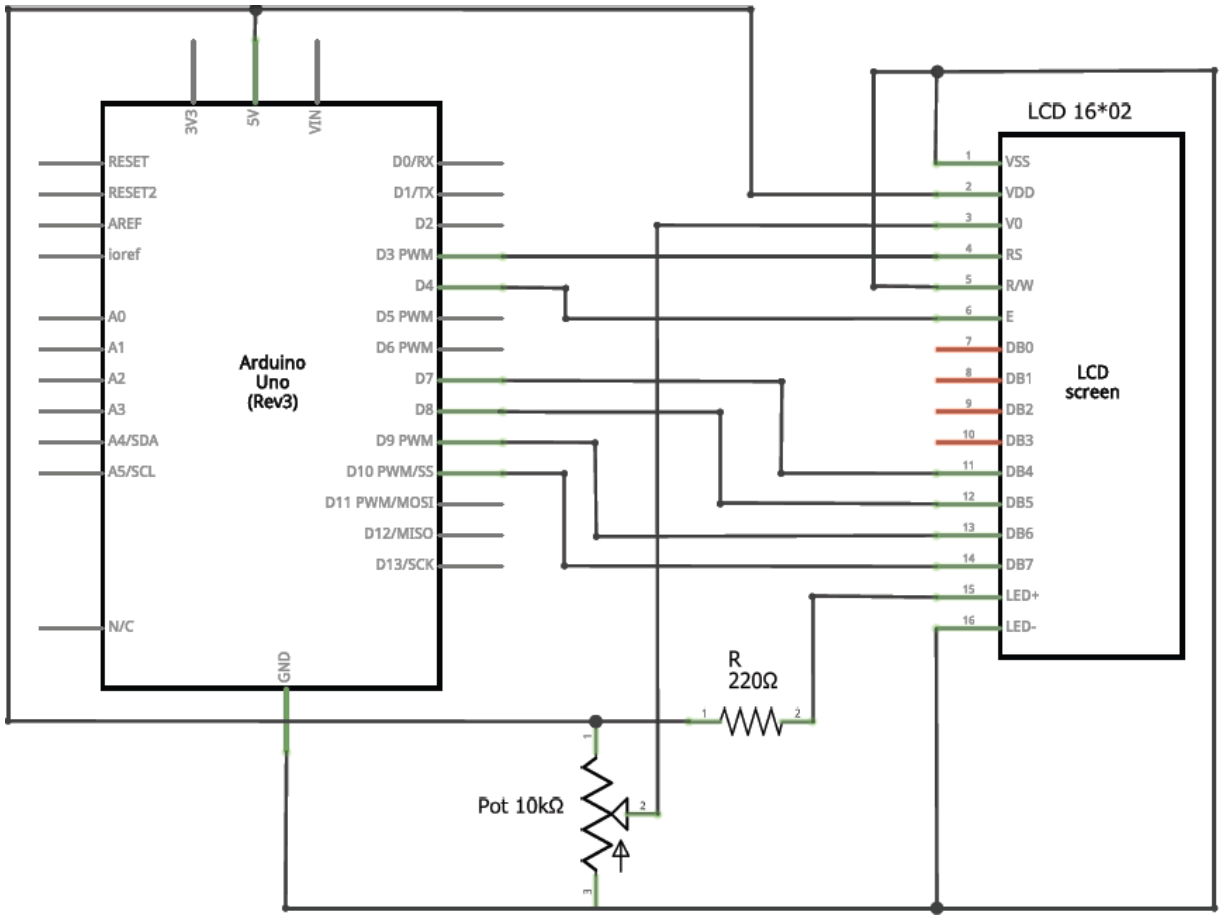
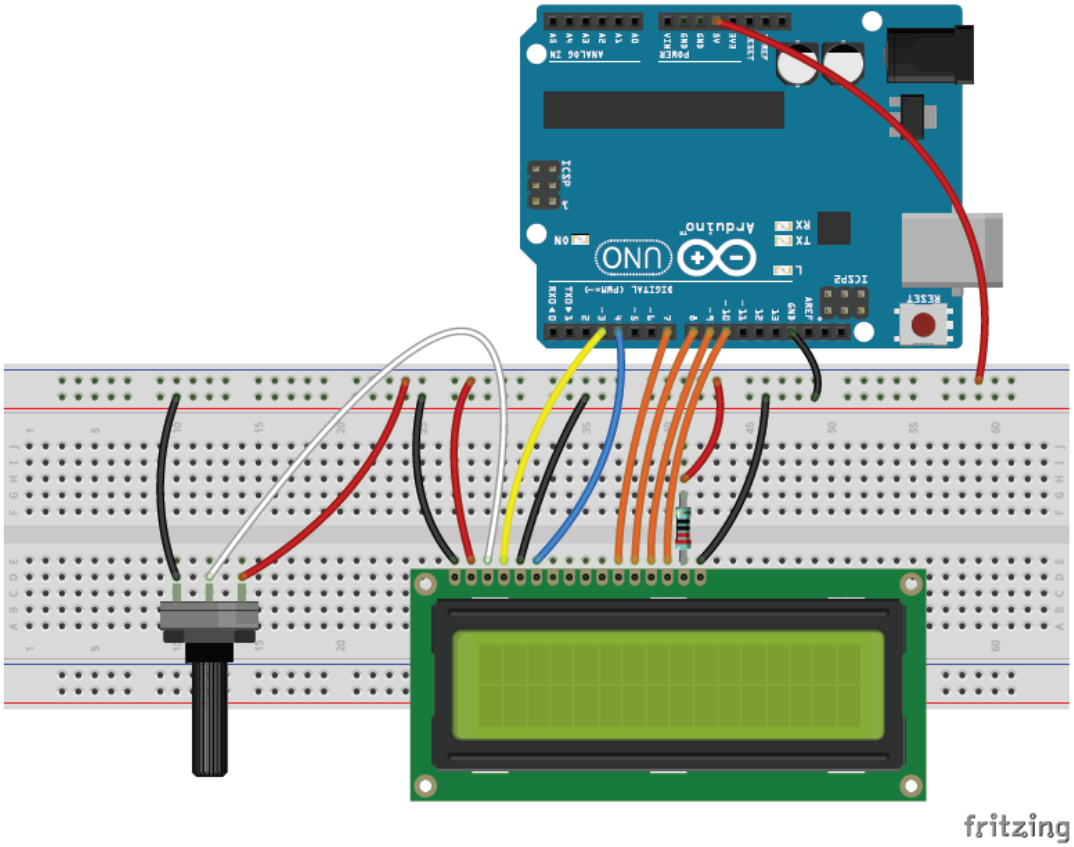


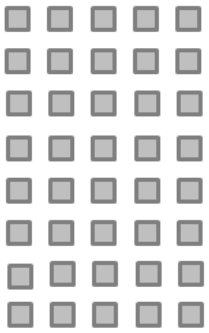
Schéma du montage



Réalisation du montage avec fritzing

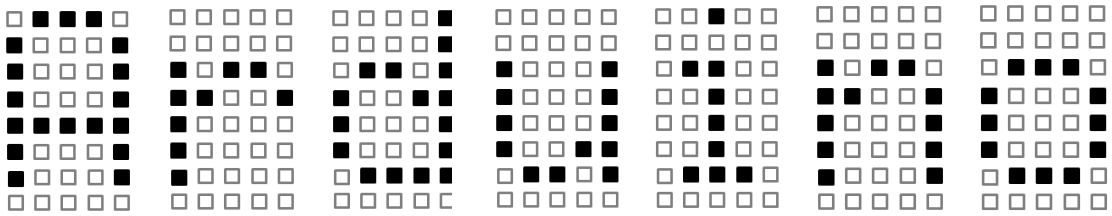
Principes importants :

Nous avons déjà dit que les différents caractères d'un afficheur étaient composés à partir d'une matrice de points (Dot-Matrix). 'Dot' signifie point, c'est le plus petit élément représentable dans cette matrice. Tout caractère est construit avec une matrice de points 5*8 (5 colonnes et 8 lignes).



Matrice de points 5*8 de l'afficheur LCD

Un emploi approprié des différents points permet de générer les caractères les plus divers. La figure suivante montre le mot 'Arduino' et les différents points à partir desquels les lettres sont composées.

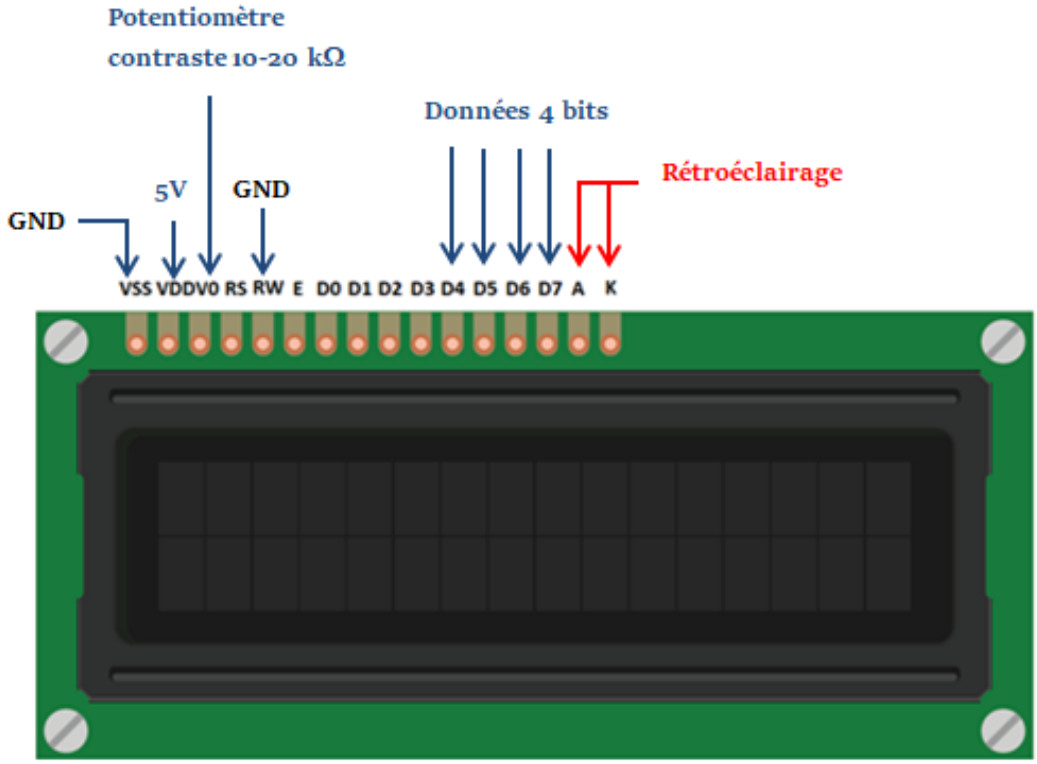


Le mot 'Arduino' composé à partir des différents points

La commande de notre afficheur LCD est parallèle, c'est-à-dire que tous les bits de données sont envoyés en même temps au contrôleur. Il existe deux modes différents (4 bites et 8bits), le mode 4 bits étant le plus utilisé parce qu'un nombre moindre de lignes de données doit être relié à l'afficheur, ce qui fait diminuer le coût.

Il existe pour cet afficheur une bibliothèque livrée avec l'IDE d'Arduino, vous pouvez raccorder un autre type d'afficheur à condition de trouver une bibliothèque appropriée.

Brochage de l'afficheur LCD :



Branchement du module LCD (16*02) à 16 broches

Sur les 8 lignes de données, seules les 4 lignes supérieures (Do àD4) sont nécessaires. La figure précédente montre le brochage du module LCD 16*02 à 16 broches.

Connexion des 16 broches de l'afficheur LCD :

VSS	Masse
VDD	Tension de service 5V
Vo	Réglage du contraste par un potentiomètre de 10 à 20 kΩ
RS	Register Select
RW	Read/Write (HIGH : Read / LOW : Write)
E	Enable
D4	Ligne de données 4
D5	Ligne de données 5
D6	Ligne de données 6
D7	Ligne de données 7
A	Anode (+) à relier à 5v via résistance 220Ω
k	Cathode (-) à relier à la masse

Remarque : Avec certains types d'afficheurs LCD, on peut brancher le rétroéclairage sur +5V sans résistance série, avec d'autres, une résistance dimensionnée en conséquence est nécessaire. Avant de connecter les deux broches (A et K) du rétroéclairage, il faut consulter la documentation du constructeur de l'afficheur utilisé.

- **Code du sketch :**

Le but de ce sketch est de faire apparaitre sur l'écran du module LCD la phrase 'Bonjour Arduino'.

Pour utiliser l'afficher d'une manière très simple, nous allons nous servir de la bibliothèque appelée « **LiquidCrystal** » existante sur l'IDE Arduino.

```
#include <LiquidCrystal.h>
#define RS 3   // Register Select
#define E   4   // Enable
#define D4  7   // Ligne de données 4
#define D5  8   // Ligne de données 5
#define D6  9   // Ligne de données 6
#define D7  10   // Ligne de données 7
#define COL  16 // Nombre de colonnes
#define LIGNES  2  // Nombre de lignes
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); // Instanciation de l'objet
void setup(){
lcd.begin(COL, LIGNES);       // Nombres de colonnes et de lignes
lcd.print("   Bonjour   "); // Affichage du texte '   Bonjour   '
lcd.setCursor(0, 1);         // Passer à la 2eme ligne
lcd.print("  Arduino Uno  "); // Affichage du texte 'Arduino Uno'
}
void loop(){  } // Fonction loop vide
```

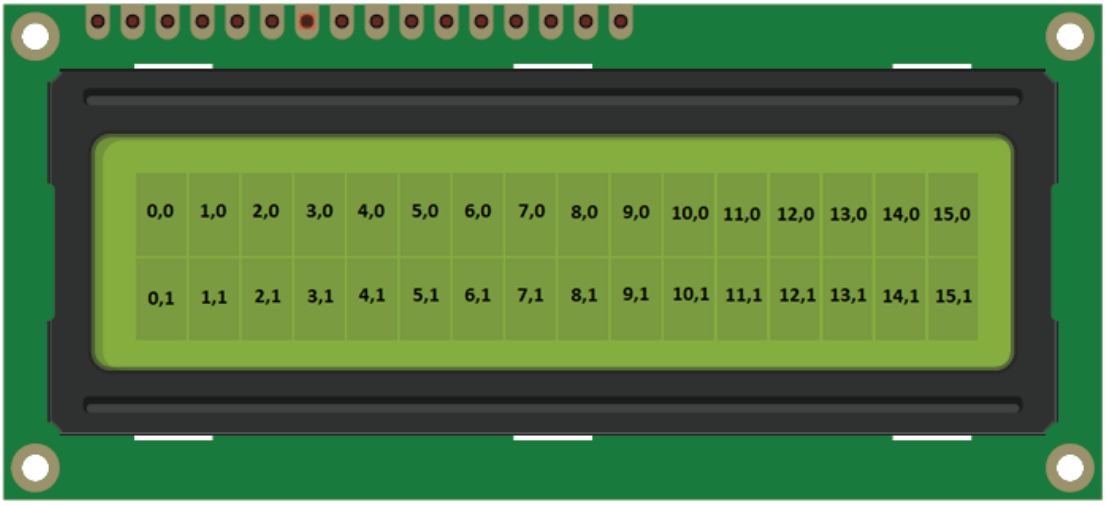
- **Explications et analyse du sketch :**

La bibliothèque **«LiquidCrystal »** doit être inclut afin de pouvoir utiliser la fonctionnalité des commandes de l'afficher LCD.

L'instruction `LiquidCrystal lcd(RS, E, D4, D5, D6, D7);` sert à instancier un objet lcd. Les paramètres : broche Register Select (RS), broche Enable et les broches des lignes de données D4 à D7 doivent être communiqués au constructeur pour générer l'objet lcd.

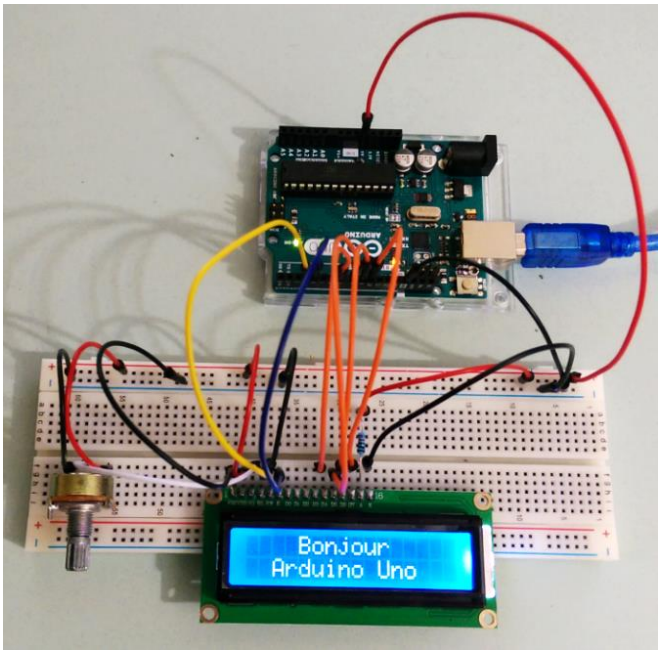
Ensuite, sur la ligne `lcd.begin(COLS, ROWS);` la méthode begin de la classe LiquidCrystal permet de communiquer le nombre de lignes et de colonnes à l'objet LCD. Tout est alors prêt maintenant, pour envoyer un texte. Puis la méthode print indique à l'objet LCD ce qui doit être affiché sur l'écran.

Remarque : Quand aucune indication n'est donnée sur la position du texte à afficher, celui-ci se place automatiquement sur la première ligne en écrasant l'ancien texte s'il existait. Pour remédier à ça, la méthode setCursor permet de positionner le curseur à l'endroit où le texte doit commencer. La figure suivante montre les différentes coordonnées des colonnes-lignes traitables par la méthode setCuror.



Coordonnées des différentes colonnes accessibles par la méthode setCursor

Nous pouvons aussi tout effacer, jusqu'au dernier caractère avec la méthode clear(), celle-ci qui n'a pas de paramètres, efface tous les caractères de l'afficheur et positionne le curseur sur la coordonné 0,0.



Réalisation du montage

LEÇON 3

Le clavier numérique (Keypad)

Au sommaire de cette 3ème leçon :

Le clavier numérique :

Les claviers numériques sont un excellent moyen permettant aux utilisateurs d'interagir avec leur projet. Vous pouvez les utiliser pour naviguer dans les menus, entrer des mots de passe ou bien contrôler des jeux et des robots.

Un clavier numérique est constitué d'un ensemble de touches ou bien de boutons poussoirs qui sont réunis en une matrice.

Clavier à membrane de matrice 4*4

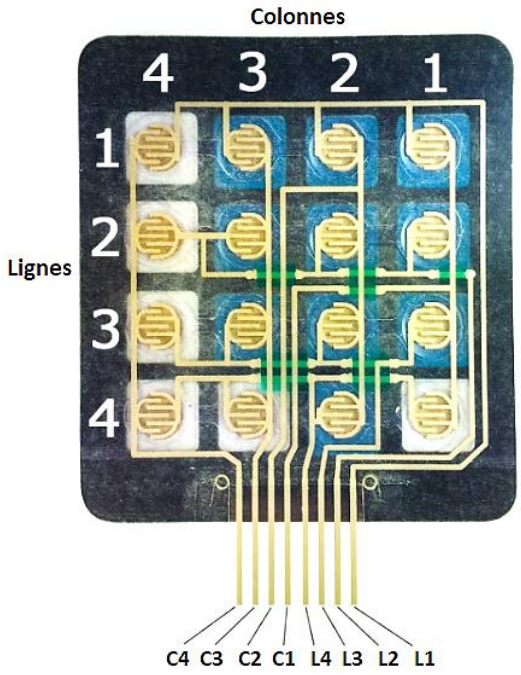
Clavier téléphonique de matrice 4*4

- **Comment un clavier numérique fonctionne-il ?**

Dans cette leçon, nous allons pratiquer sur un keypad à membrane de matrice 4*4 (figure à droite). Les touches sont alors disposées en 4 lignes et 4 colonnes.

Sous chaque touche se trouve un interrupteur à membrane. A l'intérieur du clavier, Chaque interrupteur d'une ligne est connecté d'un côté aux autres interrupteurs de la même ligne par un trait conducteur.

Keypad à membrane utilisé



Connexion entre les différents interrupteurs à l'intérieur d'un keypad à membrane 4*4

Chaque interrupteur dans une colonne est connecté de la même manière - un côté de l'interrupteur est connecté aux autres interrupteurs de cette colonne par un trait conducteur. Chaque ligne et chaque colonne est amenée vers une seule broche de sortie, ce qui donne un total de 8 broches sur un clavier 4*4.

Appuyer sur une touche ferme l'interrupteur entre un trait de colonne et un trait de ligne, permettant au courant de circuler entre une broche de colonne et une broche de ligne. Le schéma suivant d'un keypad 4*4 montre comment les lignes et les colonnes sont connectées:

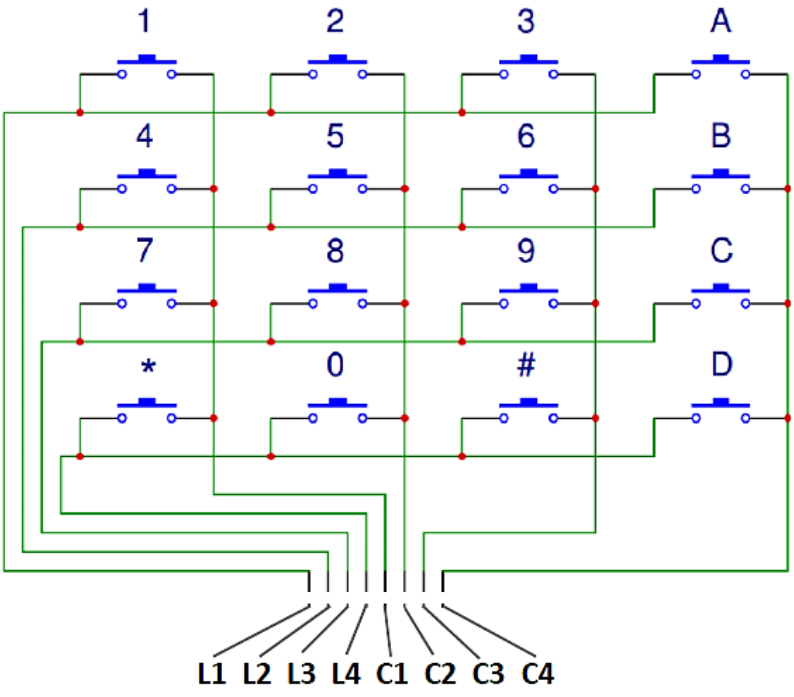


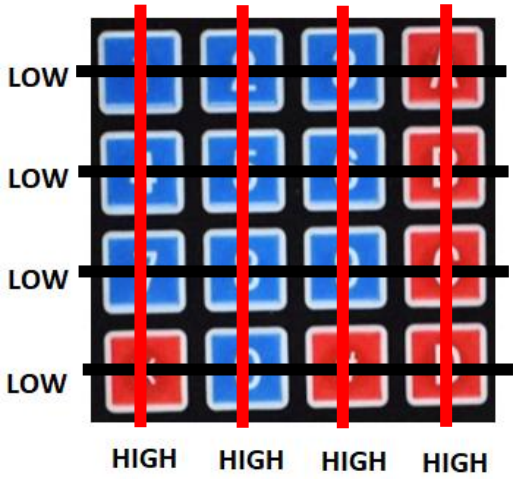
Schéma simplifié d'un keypad 4*4

Principe de détection d'une touche appuyée :

On peut détecter une touche enfoncée en lisant le niveau sur la broche de ligne et de colonne liée à la touche. Cela se passe en quatre étapes :

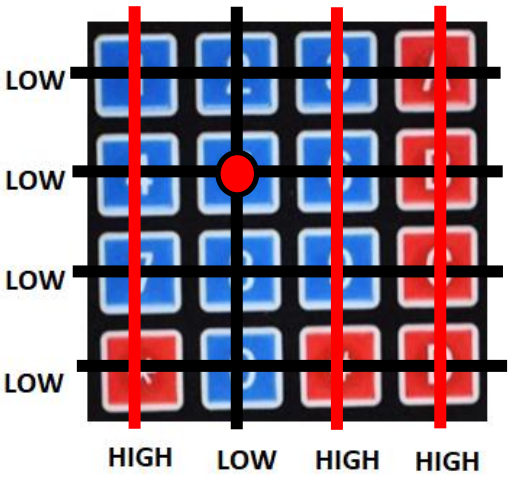
1- Tout d'abord, lorsque vous n'appuyez sur aucune touche, toutes les broches-colonnes (notées **C**) sont maintenues à l'état HIGH et toutes les broches-lignes (notées **L**) sont maintenues à l'état LOW:

2-



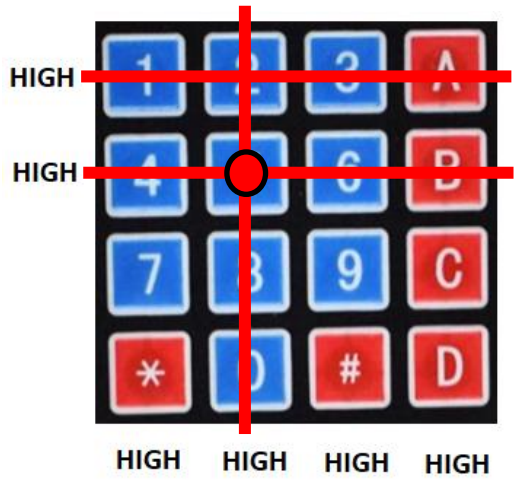
Etat des broches de lignes et de colonnes lorsque aucune touche n'est enfoncée

3- Si on détecte un niveau LOW sur l'une des colonnes (c2 par exemple), alors une pression sur une touche a été effectuée dans cette colonne. C'est parce que l'action sur l'interrupteur de la touche relie la colonne C2 avec la ligne R2. Par conséquent C2 est tirée vers le niveau LOW:



Changement d'état de la broche liée à la colonne 2 lorsque la touche 5 est appuyée

4- Maintenant, on sait la colonne où se trouve la touche, Il faut juste détecter la ligne sur laquelle elle se situe. Cette opération peut être faite en commençant à mettre des niveaux HIGH sur les lignes de façon séquentielle (l'une après l'autre) et de vérifier si C2 redevient à l'état HIGH. La logique est que si une touche dans cette ligne a été pressée, alors la valeur écrite sur cette ligne sera reflétée dans la colonne correspondante (C2) car ils sont court-circuités.



Détection de la broche de ligne liée à la touche enfoncée en mettant chaque broche-ligne à l'état HIGH

Lorsque la broche de colonne C2 redevient à l'état HIGH, on détecte que la broche de ligne de la touche enfoncée est L2. La position de la clé dans la matrice est (2,2)

Une fois que cela est détecté, c'est à nous de le nommer ou de lui fournir une tâche sur l'événement de la touche. On n'a pas besoin de programmer tout ça pour détecter une touche appuyée, Nous servirons dans nos sketch une bibliothèque spéciale dédiée au clavier numérique.

- **L'affichage d'une touche appuyée sur le moniteur série de l'IDE Arduino :**

Pour une démonstration de base de la configuration du clavier, nous allons voir comment imprimer chaque touche appuyée d'un keypad 4*4 sur le moniteur série.

Matériels et composants nécessaires :

Aduino Uno

Keypad à membrane 4*4

Fils de connexion

Schéma du montage :

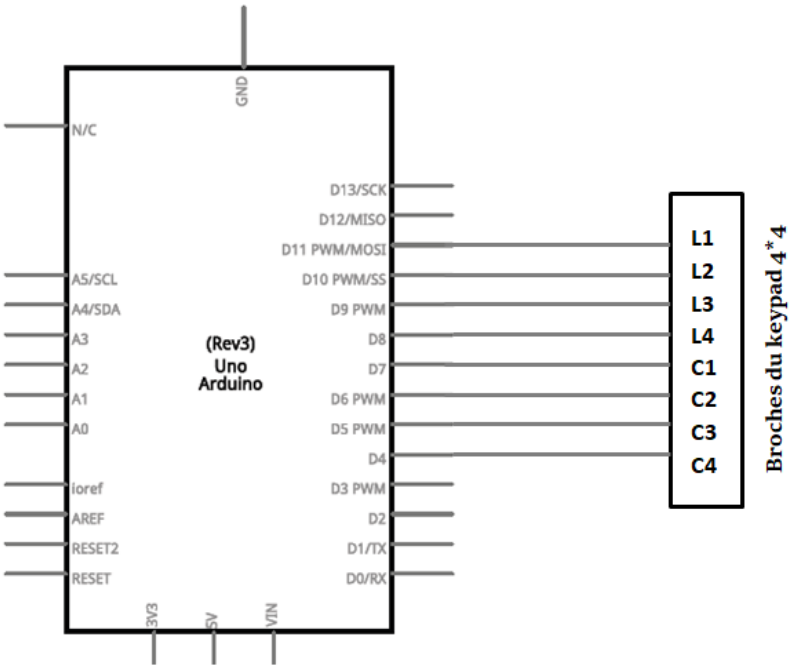
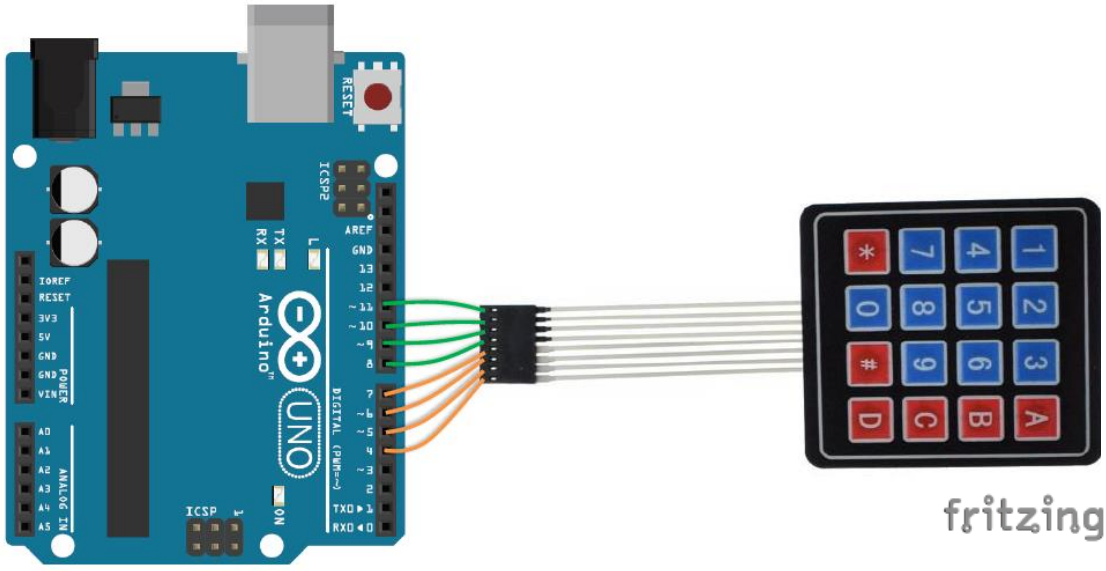


Schéma du montage



Réalisation du montage avec fritzing

Code du sketch :

Nous utiliserons la bibliothèque 'keypad'. Cette bibliothèque s'occupe de la configuration des broches et de l'interrogation des différentes colonnes et lignes. Pour installer cette bibliothèque, Rendez-vous sur la barre de menu de l'IDE Arduino, puis accédez à **Croquis** > **Inclure une bibliothèque** > **Gérer les bibliothèques** et cherchez "keypad". Cliquez sur la bibliothèque, puis cliquez sur **installer**.

```
#include <Keypad.h>
const byte LIGNES = 4; //Nombre de lignes
const byte COL = 4;   //Nombre de colonnes
char Touches[LIGNES][COL] = {  //Tableau contenant les touches de notre keypad
 {'1', '2', '3', 'A'},
 {'4', '5', '6', 'B'},
 {'7', '8', '9', 'C'},
 {'*', '0', '#', 'D'}
};
byte BrochesLignes[LIGNES] = {6, 7, 8, 9}; //Broches-lignes connectées à L1,L2,L3 et L4
byte BrochesCol[COL] = {10, 11, 12, 13}; //Broches-colonnes connectées à C1,C2,C3 et C4
Keypad  MonKeypad = Keypad(makeKeymap(Touches), BrochesLignes, BrochesCol, LIGNES,
COL); //Instanciation de l'objet

void setup(){
Serial.begin(9600); //définition de le vitesse de transmission (baud)
}

void loop(){
 char ToucheAppuyee = MonKeypad.getKey();   //Lecture de la touche appuyée
 if (ToucheAppuyee){           //Une touche quelconque a-t-elle été appuyée?
 Serial.println(ToucheAppuyee);    //Afficher la touche appuyée sur le moniteur série
  }
}
```

Explication et analyse du sketch :

La première ligne incorpore la bibliothèque 'keypad' afin de pouvoir exploiter les fonctionnalités de commande du clavier numérique.

Ensuite nous avons défini le nombre de lignes et de colonnes de notre clavier, ainsi que les différents caractères qui seront imprimés lorsqu'une touche sur le keypad est appuyée.

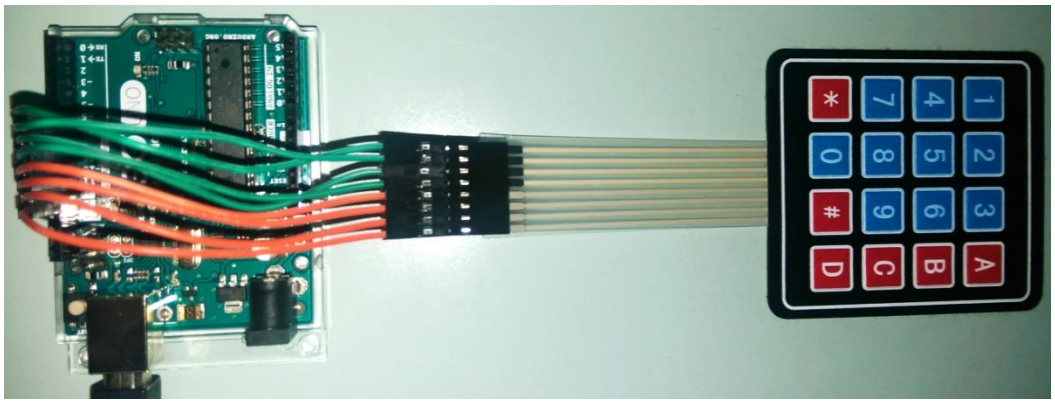
Les deux tableaux 'BrochesLignes[LIGNES]' et 'BrochesCol[COL]' sont initialisés avec les numéros des broches de connexion au lignes et aux colonnes du keypad.

La ligne :

```
Keypad MonKeypad = Keypad(makeKeymap(Touches), BrochesLignes, BrochesCol, LIGNES, COL); //Instanciation de l'objet
```

génère l'instance MonKeypad de la classe Keypad qui est définie dans la bibliothèque, et transmet les trois tableaux et le nombre de lignes et de colonnes au constructeur de la classe Keypad. Ces informations les sont nécessaires pour commencer à évaluer sur laquelle des 16 touches on a appuyé.

La ligne : `char ToucheAppuyee = MonKeypad.getKey();` affecte le résultat renvoyé par la méthode getKey à la variable ToucheAppuyee du type char. Si getKey renvoie un caractère, ça veut dire qu'une touche a été appuyée, alors on affiche le caractère de la touche ainsi appuyée sur le moniteur série : `Serial.println(ToucheAppuyee);`



Réalisation du montage

Application : Accès sécurisé avec mot de passe

Nous allons essayer dans cette partie, d'intégrer les LED et l'afficheur LCD étudiées précédemment dans une application, tout en se basant sur un clavier numérique.

Pour vérifier le bon fonctionnement du montage de cette application, fixons nous les spécifications suivantes :

- Lorsqu'on alimente le montage, un mot de passe à entrer via le keypad, doit être demandé sur l'afficheur LCD avec le message "**Mot de passe :** ". Trois LED : Rouge, vertes et bleu sont à l'état OFF au début.
- Si le mot de passe entré est correct, un message "**Valide ^_^ Acces**" s'affiche sur le LCD et la LED verte s'allume pendant une durée de 2 secondes et redevient à son état OFF, juste après, la LED bleu s'allume et reste allumée avec un message "**Appyez sur 'A'** pour continuer " qui s'affiche pour informer l'utilisateur s'il veut reprendre dès le début. Si la touche 'A' est appuyé, la LED bleu s'éteint, et le fonctionnement recommence à zéro.
- Si le mot de passe tapé est incorrect, un message **"Non valide ! Ressayez "** s'affiche sur le LCD et la LED rouge s'allume et reste allumée jusqu'à ce que le mot de passe soit vérifié correcte, à ce moment-là elle redeviendra à son état OFF, on demande encore et encore à l'utilisateur d'entrer le mot de passe.

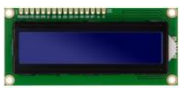
Matériels et composants nécessaires :

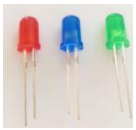

Aduino Uno

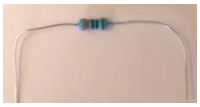

Breadboard

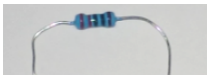

Keypad à membrane 4*4


Fils de connexion

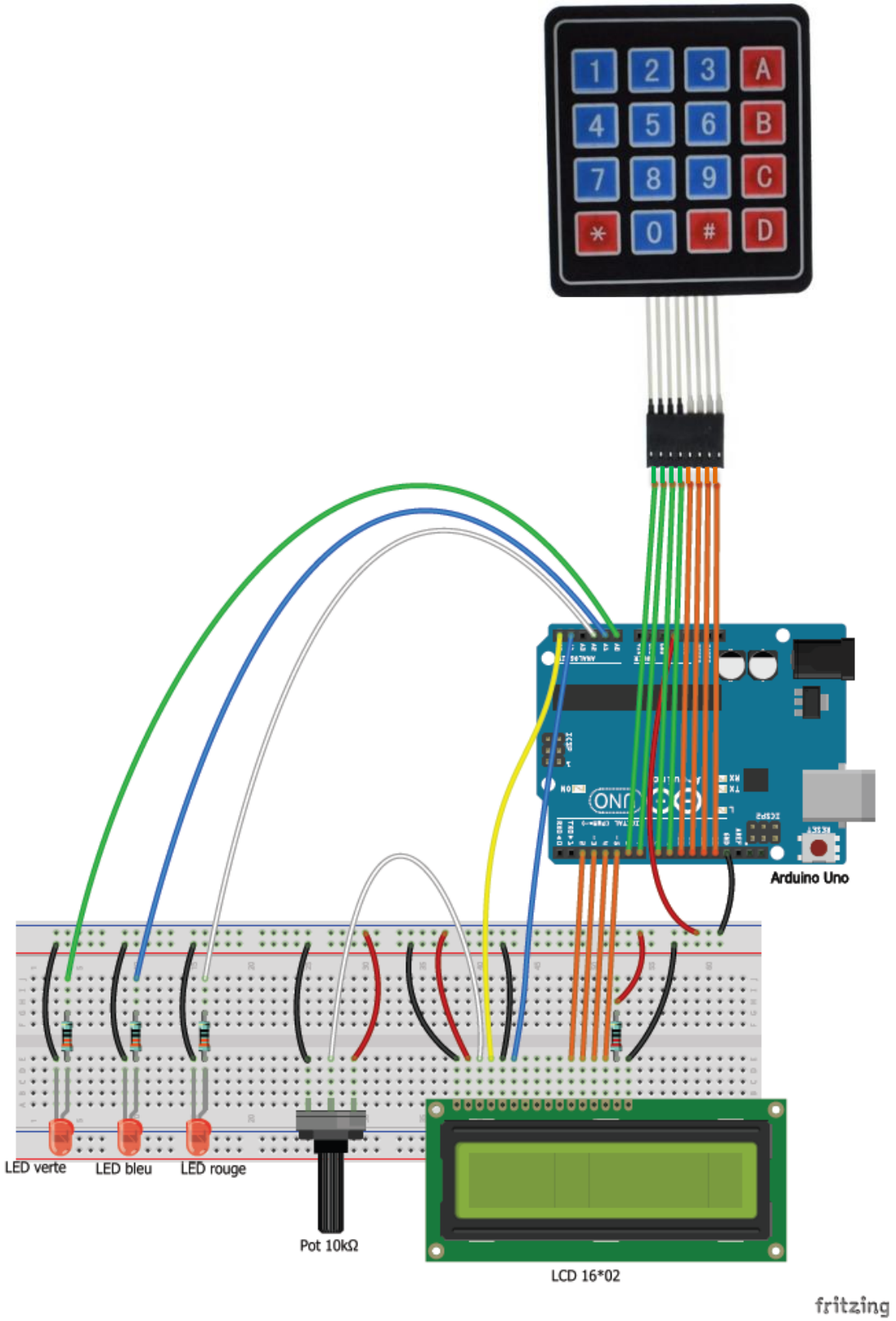

Afficheur LCD 16*02 avec interface standard 16 broches


3 * LED


3 * Résistance 330 Ω


Résistance 220 Ω

- **Montage de l'application :**



Réalisation du montage avec fritzing

- **Code du sketch :**

```
/*1- Incorporation des bibliothèques, définition des constantes et déclaration des variables*/
#include <LiquidCrystal.h>
#include <Keypad.h>
#define RS A5   // Register Select
#define E  A4   // Enable
#define D4 2    // Ligne de données 4
#define D5 3    // Ligne de données 5
#define D6 4    // Ligne de données 6
#define D7 5    // Ligne de données 7
#define COLLCD  16 // Nombre de colonnes
#define LIGNESLCD  2  // Nombre de lignes
#define TailleMdp 5   // Taille du mot de passe
int LedVerte = A0;    // Broche liée à la LED verte
int LedBleu  = A1;    // Broche liée à la LED bleu
int LedRouge = A2;    // Broche liée à la LED rouge
char Donnee[TailleMdp];  //Tableau pour stocker les caractères donnée
char Mdp[TailleMdp] = "0000"; //Tableau mémorisant les caractères du mot de passe
byte comptDonnee = 0; //compteur donnée
char ToucheAppyee;
char continuer=0;
const byte LIGNESPAD = 4; //Nombre de lignes DU KEYPAD
const byte COLPAD = 4;    //Nombre de colonnes
char Touches[LIGNESPAD][COLPAD] = {  //Tableau contenant les caractères des touches de
//notre keypad
 {'1', '2', '3', 'A'},
 {'4', '5', '6', 'B'},
 {'7', '8', '9', 'C'},
 {'*', '0', '#', 'D'}
};
byte BrochesLignes[LIGNESPAD] = {6, 7, 8, 9}; //Broches-lignes connectées à L1,L2,L3 et L4
```

```
byte BrochesCol[COLPAD] = {10, 11, 12, 13}; //Broches-colonnes connectées à C1,C2,C3 et C4
/*2- Instanciation des objets*/
LiquidCrystal lcd(RS, E, D4, D5, D6, D7); // Instanciation de l'objet lcd
Keypad  MonKeypad = Keypad(makeKeymap(Touches), BrochesLignes, BrochesCol,
LIGNESPAD, COLPAD); //Instanciation de l'objet MonKeypad
/*3-programmation des différentes broches utilisées en sorties et définition du nombre de
lignes et de colonnes de l'afficheur LCD*/

void setup(){
lcd.begin(COLLCD, LIGNESLCD);        // Nombres de colonnes et de lignes de l'afficheur LCD
pinMode(LedRouge, OUTPUT);   // Programmation de la broche A2 comme sortie numérique
pinMode(LedVerte, OUTPUT);   // Programmation de la broche A0 comme sortie numérique
pinMode(LedBleu, OUTPUT);    // Programmation de la broche A1 comme sortie numérique
}

void loop(){
/*4- Programme traitant l'entrée de la donnée par l'utilisateur*/
lcd.setCursor(0,0); //Placer le curseur sur la colonne 0 de la ligne 1
lcd.print(" Mot de passe : "); // Afficher le message " Mot de passe : "
ToucheAppyee = MonKeypad.getKey(); // Affectation du résultat de la lecture à la variable
//'ToucheAppyee'
if (ToucheAppyee){  // Si une touche est appuyée
Donnee[comptDonnee] = ToucheAppyee; //stocker le caractère de la touche appuyée dans le
// tableau 'Donnee'
lcd.setCursor(comptDonnee+6,1); // Centrer l'affichage du mot de passe
lcd.print("*"); // Afficher un étoile pour masquer le mot de passe entré
comptDonnee++;  // Incrémenter le compteur donnée
  }
/*5-Vérifier si la taille de la donnée entrée par l'utilisateur est égale à celle du mot de passe
(5)*/
if(comptDonnee == TailleMdp-1){ //Vérifier si le nombre de caractères de la donnée est
//égale à celui du mot de passe
```

```
delay(200); // Attendre 200ms
lcd.clear(); //Efacer l'afficheur LCD
/*6-programme traitant le cas où un mot de passe valide est entré par l'utilisateur*/
if(strcmp(Donnee, Mdp)==0) {    //Comparer le contenu du tableau Donnee avec celui du
//tableau Mdp (0 s'ils sont égaux)
digitalWrite(LedRouge, LOW); // Eteindre la LED rouge
lcd.setCursor(3,0);
lcd.print("Valide ^_^");
lcd.setCursor(4,1);
lcd.print("Acces");
digitalWrite(LedVerte, HIGH);// Allumer la LED verte
delay(2000); //Attendre 2 secondes
digitalWrite(LedVerte, LOW);// Eteindre la LED verte
digitalWrite(LedBleu, HIGH);// Allumer la LED bleu
lcd.clear();
while(continuer !='A'){ // Tant que la variable 'continuer' est différente de la lettre 'A'
lcd.print("Appuyez sur 'A'");
lcd.setCursor(0,1);
 lcd.print("pour repredre   ");
 continuer=MonKeypad.getKey(); // Affectation du résultat de la lecture à la variable
//'continuer'
}
digitalWrite(LedBleu, LOW);// Eteindre la LED bleu
continuer=0;
 }
/*7-programme traitant le cas où un mot de passe non valide est entré par l'utilisateur*/
else {
lcd.print("   Non valide !   ");
lcd.setCursor(0,1);
lcd.print("    Ressayez    ");
digitalWrite(LedRouge, HIGH);// Allumer la LED rouge
delay(2000);    //Attendre 2 secondes
```

```
 }
/*8-Effacer l'afficheur LCD et supprimer l'ancienne donnée entrée par l'utilisateur */
lcd.clear();
SupDonnee();  //Appelle de la fonction de suppression de la donnée
 }
}
/*9-Définition de la fonction de suppression de l'ancienne donnée*/
void SupDonnee() {
while(comptDonnee !=0){       //Tant que le compteur donnée est différent de 0
Donnee[comptDonnee--] = 0;  //Effacer l'élément numéro (comptDonnee) et décrémenter
//comptDonnee
 }
}
```

- **Explication et analyse du sketch :**

Le code du sketch peut être divisé en 9 parties :

1- Cette première partie, concerne l'incorporation des deux bibliothèques «LiquidCrystal » et « Keypad », la définition des différentes constantes et la déclaration et initialisation des variables qui seront utilisé dans le sketch.

2- Dans la 2ème partie, nous avons instancié les deux objets : l'objet lcd de la classe **LiqudCrystal** et l'objet MonKeypad de la classe **Keypad**.

3- La 3ème partie consiste à programmer les différentes broches utilisées en sorties numériques, ainsi que la définition du nombre de lignes et de colonnes de l'afficheur LCD.

4- Cette partie du code, traite la saisie du mot de passe (donnée) par un utilisateur : il s'agit de stocker chaque caractère entré par l'utilisateur dans la case numéro '**comptDonnee**' du tableau appelé '**Donnee**' et d'incrémenter '**comptDonnee**' pour passer à la case suivante. La méthode getKey() renvoi le caractère appuyé sinon elle n'envoie rien.

5- A chaque saisie d'un caractère par l'utilisateur en appuyant sur une touche, on vérifie si le nombre de caractères saisis en totale est égale à celui du mot de passe (4 dans ce

cas). La taille du mot de passe est déclarée 5, mais n'oubliez pas que le dernier élément d'une chaine de caractères est toujours réservé pour le caractère nul '\0'.

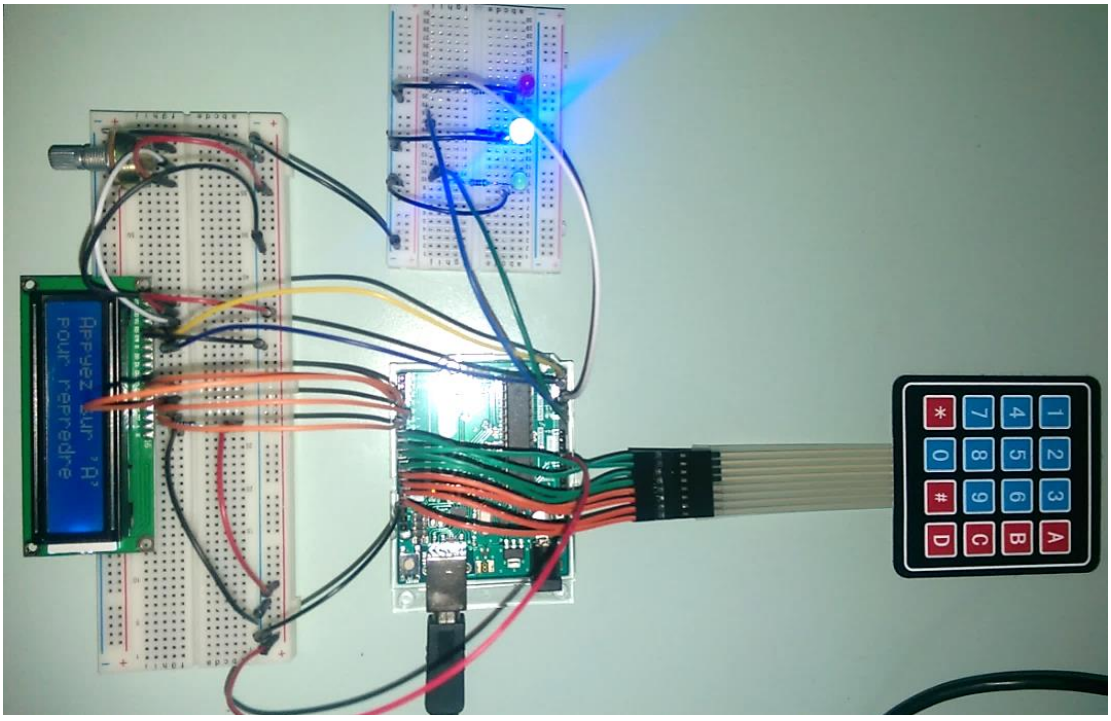
Dans le cas où les deux nombres sont égaux, on peut alors comparer la donnée saisie avec le mot de passe, on efface d'abord l'afficheur LCD.

6- Cette partie se charge à traiter le cas où l'utilisateur saisie un mot de passe correcte. Dans laquelle on compare les deux chaines '**Donnee**' et '**Mdp**' à l'aide de la fonction strcmp, si les deux chaines sont égales, cette fonction renvoie zéro. On exécute les actions décrite dans le cahier des charges et on demande à l'utilisateur de taper le caractère 'A' pour recommencer.

7- Le cas d'un mot de passe erroné tapé par l'utilisateur, est traité dans cette partie du code. On informe l'utilisateur avec un message on lui demandant de retaper la clé à nouveau et on allume une LED rouge de signalisation.

8- Cette partie se charge à effacer l'afficheur LCD et à formater l'ancienne chaine-donnée saisie par l'utilisateur, pour lui transférer les nouveaux caractères la prochaine fois.

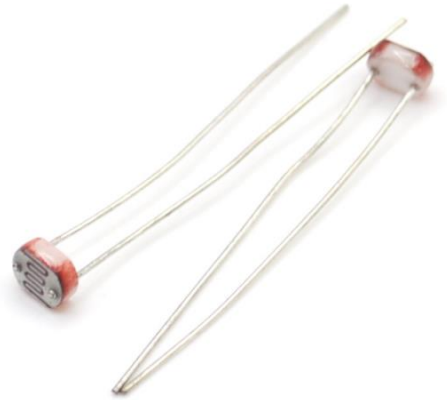
9- La fonction de formatage est définie dans cette dernière partie, elle consiste à parcourir et supprimer chaque caractère de l'ancienne chaine–donnée avec l'indice **comptDonnee,** tout en commençant par le dernier caractère et arrivant au premier, raison pour laquelle **comptDonnee** se décrémente à chaque fois.



Réalisation du montage

LEÇON 4

Le détecteur de lumière

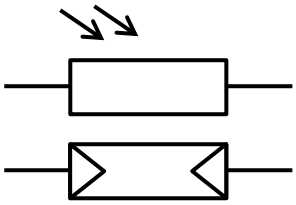


Au sommaire de cette 4ème leçon :

Le détecteur de lumière (photorésistance) :

Une photorésistance, également appelée LDR (**L**ight **D**epending **R**esistor), est un composant électronique dont la résistivité varie en fonction de l'intensité lumineuse qui lui est appliquée. Plus une LDR est éclairée, plus sa résistivité baisse.

Photorésistance (LDR)



Symbole électronique d'une LDR

Ce type de LDR se caractérise principalement par sa facilité de mise en œuvre, son faible coût et sa sensibilité élevée. Les domaines d'application d'une LDR sont variés, parmi lesquels on peut citer :

- Commande de l'éclairage publique ou domestique lorsque la luminosité baisse.
- Mesure de la luminosité extérieure dans les appareils photographiques.
- Contrôle d'accès à des zones sécurisées avec des barrières lumineuses.

La plage de résistance d'une LDR dépend du matériau utilisé, et présente approximativement une résistance : $1 \leq \mathbf{R} \leq 10$ MΩ, une intensité d'éclairement de 1000 lux (lx) environ, génère une résistance comprise entre 75 et 300 Ω.

Application : Commande de l'intensité de lumière d'une LED

Dans cette application, nous allons réaliser un circuit capable de réagir à des influences extérieures. On compte faire varier l'intensité de lumière d'une LED en fonction de l'éclairement extérieure mesuré avec une LDR. Les spécifications du fonctionnement sont :

- Plus une quantité de lumière atteignant la LDR, plus l'intensité de lumière de la LED est faible.
- Le pourcentage de l'éclairement extérieur est affiché sur un module LCD.

Matériels et composants nécessaires :

Aduino Uno

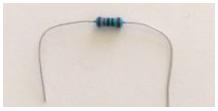
Breadboard

Photorésistan

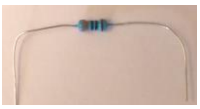
Fils de connexion

LED

Afficheur LCD 16*02 avec interface standard 16 broches

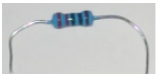


Résistance 10 kΩ



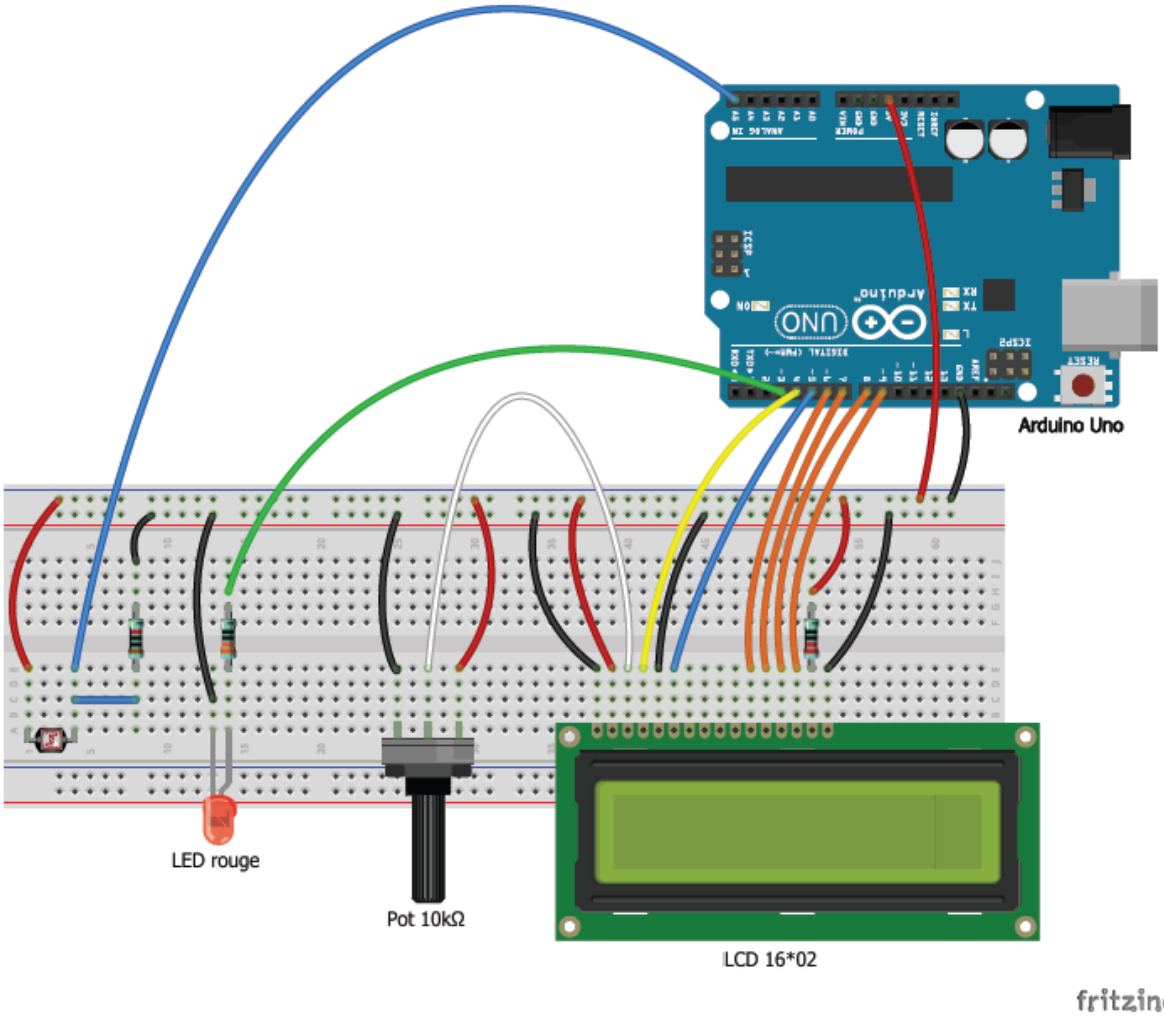
Résistance 330 Ω

Potentiomètre 10 KΩ



Résistance 220 Ω

- **Montage de l'application :**



Réalisation du montage avec fritzing

- **Code du sketch :**

```
/* Incorporation de la bibliothèque, définition des constantes et déclaration et initialisation des variables*/
#include <LiquidCrystal.h>
#define COL 16
#define LIGNES 2
#define RS 4      // Register Select
#define E 5       // Enable
#define D4 6    // Ligne de données 4
#define D5 7    // Ligne de données 5
#define D6 8   // Ligne de données 6
#define D7 9  // Ligne de données 7
int BrocheLed = 3; // LED connectée à la sortie numérique 3 (PWM)
int EntreeAnalog = 5; // Entrée analogique 5 liée à la sortie de la LDR
int val = 0;   // Variable pour stocker la valeur de l'entrée analogique 5
int mapval=0;   // Variable pour stocker le résultat retourné par la fonction map
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);   // Instanciation de l'objet lcd
/* Programmation de la broche numérique 3 en sorties et définition du nombre de lignes et de colonnes de l'afficheur LCD*/
void setup() {
lcd.begin(COL, LIGNES);  // Nombres de colonnes et de lignes
pinMode(BrocheLed, OUTPUT); // Programmation de la broche 3 comme sortie
}
/* Code traitant la commande de l'intensité lumineuse de la LED et l'affichage du pourcentage de l'éclairage extérieur sur le module LCD*/
void loop() {
val = analogRead(EntreeAnalog); // Lire la valeur à l'entrée analogique 0
mapval=map(val,0,1023,0,255); //Affectation de la valeur renvoyée par la fonction map à la
//variable mapval
analogWrite(BrocheLed, 255-mapval); //
lcd.setCursor(0,0);
```

```
lcd.print("   Luminosite   ");
lcd.setCursor(0,1);
lcd.print("      ");
lcd.print((mapval*100/255));  //Afficher la valeur du pourcentage de l'éclairement
lcd.print(" %");
delay(2000);
}
```

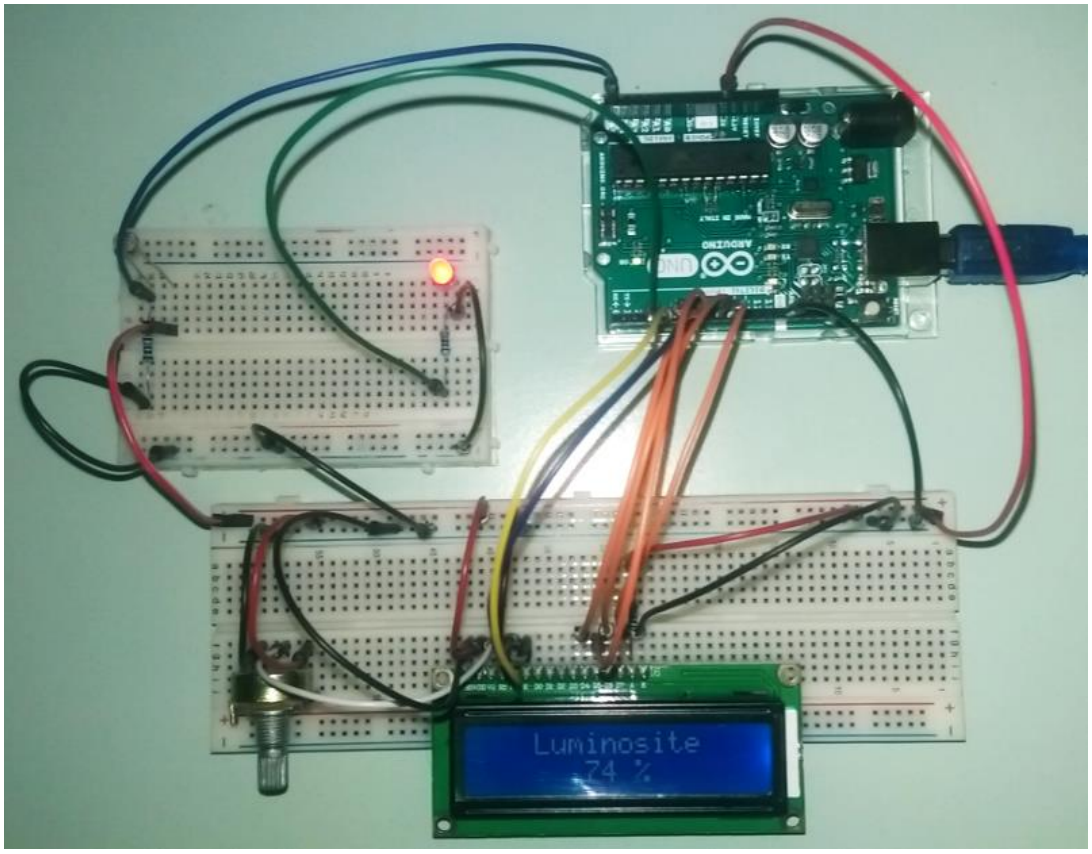
- **Explication et analyse du sketch :**

Comme d'habitude, la première partie du code est réservée pour la déclaration et l'initialisation des variables, la définition des constantes et pour l'instanciation d'un objet LCD.

Dans la fonction setup, nous avons programmé la broche numérique 3 comme sortie et on a communiqué le nombre de lignes et de colonnes à l'objet LCD avec la méthode begin.

Le code permettant de commander l'intensité lumineuse de la LED et d'afficher le pourcentage d'éclairement sur le module LCD est défini au sein de la fonction loop. Le principe est de lire d'abord le nombre de conversion de la tension délivrée par la LDR à l'entrée analogique 5 et de stocker cette valeur dans la variable 'val'. Ensuite, avec la fonction map on transpose le domaine de valeurs (0-1023) dans le domaine (0-255) (vu que 255 est la valeur maximale qu'on peut transmettre à une sortie PWM). Et on affecte la nouvelle valeur renvoyée par map à la variable 'mapval'.

Ensuite on transmet la valeur (255-mapval) à la sortie 3. De ce fait, l'intensité de lumière de la LED augmente si l'éclairement extérieur baisse. Il reste d'afficher le pourcentage de cet éclairement sur l'afficheur LCD, pour ce faire, nous avons utilisé la formule (mapval*100)/255 qui donne un résultat en %.



Réalisation du montage

LEÇON 5

Le Relais électromagnétique

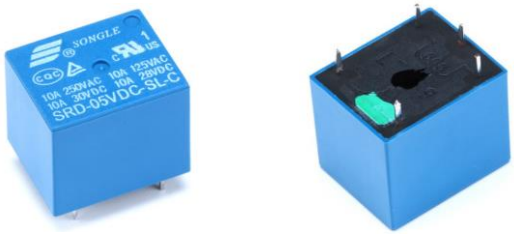
Au sommaire de cette 5ème leçon :

Avertissement

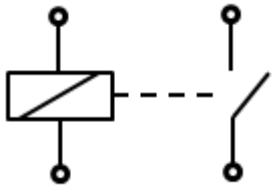
Cette leçon implique une tension de service élevée qui peut causer des blessures graves ou la mort. Veuillez prendre toutes les précautions nécessaires et débrancher toute alimentation d'un circuit avant de commencer à travailler.

Le Relais électromagnétique :

Le relais est un composant électromécanique qui est composé principalement d'un électroaimant qui, lorsqu'il est alimenté, transmet une force à un système de commutation électrique : le contact.

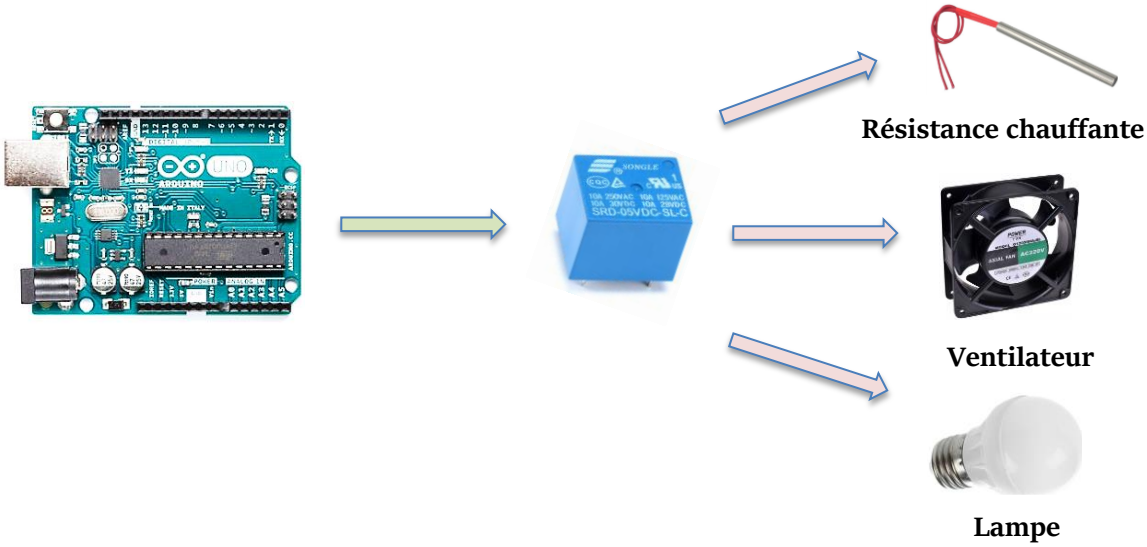


Relais électromagnétique



Symbole électrique d'un relais électromagnétique

Un relais permet l'ouverture ou bien la fermeture d'un circuit électrique par un second circuit complètement isolé (isolation galvanique). L'opération principale de ce composant est d'établir ou de rompre le contact à l'aide d'un signal sans aucune intervention humaine pour l'activer ou l'éteindre. Il est principalement utilisé pour contrôler un circuit de haute puissance en utilisant un signal de faible puissance.



Commande des appareils à 230V avec Arduino Uno à travers un relais électromagnétique

L'une des choses les plus utiles que vous pouvez faire avec un Arduino est de contrôler les appareils à basse tension (230V) comme un ventilateur, une lampe, une résistance chauffante et d'autres appareils électroménagers.

Comme la carte Arduino Uno fonctionne à très basse tension (5V), elle ne peut pas contrôler directement ces appareils, mais elle peut commander un relais 5V (à électroaimant 5V) qui commande à son tour un appareil à 230V.

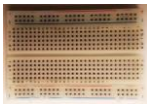
Application : Clignotement de deux lampes 230v

Nous allons voir dans cette partie, une application simple du relais électromagnétique. On va faire clignoter deux lampes 230V pendant une durée de 5 secondes chacune (une seule lampe à la fois). Nous allons utiliser un module à deux relais (figure ci-dessus) avec des optocoupleurs intégrés pour l'isolation galvanique.

Module à deux relais utilisé

Matériels et composants nécessaires :

Aduino Uno

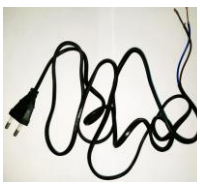


Breadboard

Fils de connexion

Module à deux relais

Deux lampes 220V

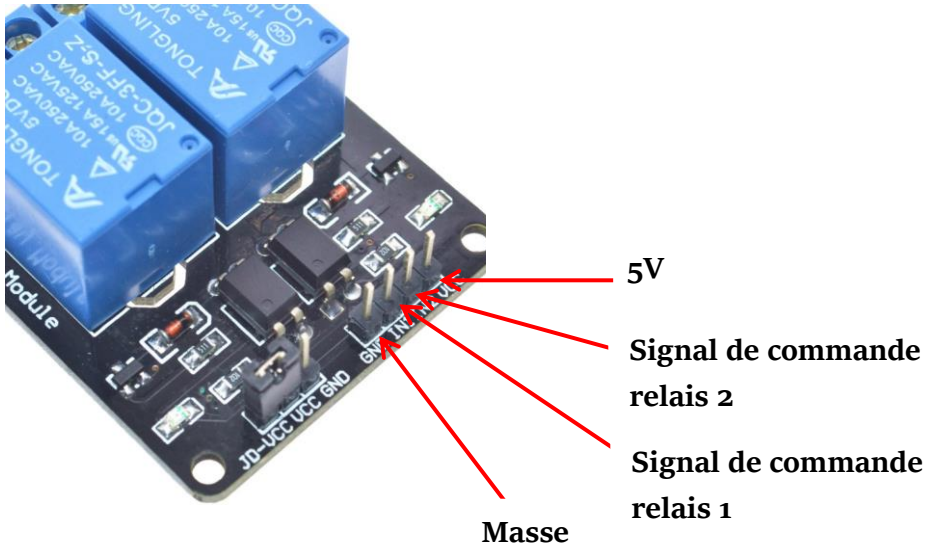


2 * Prise-câble 230V

2 * douilles pour lampes 230V

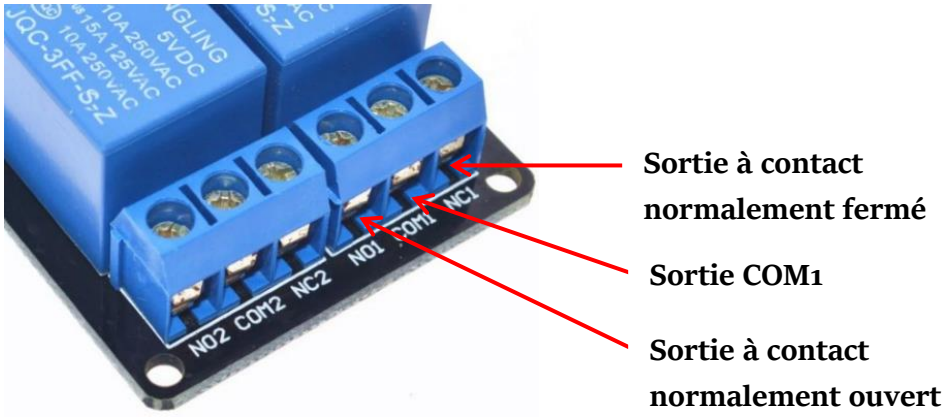
- **Brochage du module à 2 relais :**

Du coté très basse tension (5V), le module comporte 4 entrées qui peuvent être raccordées avec un circuit de commande : **VCC** (Très basse tension), **GND** (Masse), **IN1** (Signal de commande pour le relais 1), **IN2** (Signal de commande pour le relais 2).



Broches d'entrées du côté très basse tension du module à 2 relais

De l'autre côté basse tension (230v), le module est doté de 6 bornes de sorties : **NO1** (sortie à contact normalement ouvert), **NC1** (sortie à contact normalement fermé) et une sortie **COM1** à raccorder avec le fil de phase d'alimentation 230v. Même chose pour les trois autres sorties sauf qu'elles sont réservées pour le relais numero 2.



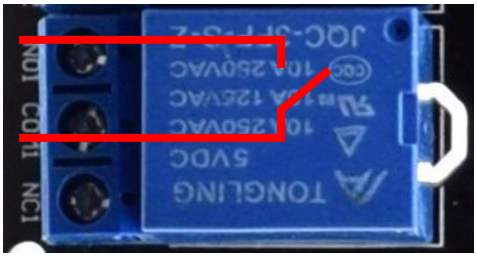
Broches de sorties du côté basse tension du module à 2 relais

Contacts NO et NC du relais :

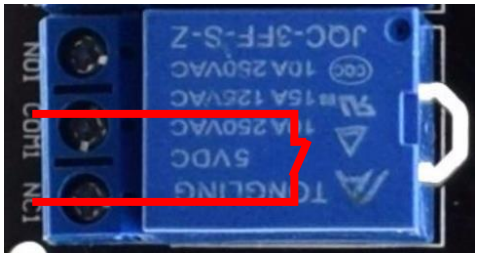
Le relais a deux types de contacts électriques à l'intérieur - normalement ouvert (**NO** : **N**ormally **O**pen) et normalement fermé (**NC** : **N**ormally **C**losed). Celui que vous utiliserez dépendra de votre application et de si vous voulez que le signal 5V ferme ou bien ouvre le contact du relais. Le courant de l'alimentation 230V entre dans le relais à la borne commune (COM) dans les deux configurations 'normalement ouvert' et 'normalement fermé'.

Le contact normalement ouvert (NO) :

Dans la configuration 'normalement ouvert' (figure ci-dessus), le contact NO1 est ouvert à l'arrêt. Lorsque le relais reçoit un signal HIGH, le contacte NO1 (230V) se ferme et permet au courant de circuler de la borne COM1 vers la borne NO1. Un signal LOW désactive le relais (ouvre le contact NO1) et arrête le courant de circuler.



Contact NO1 à l'intérieur du relais



Contact NC1 à l'intérieur du relais

Le contact normalement fermé (NC) :

Dans la configuration 'normalement fermée', un signal HIGH ouvre le contact NC1 qui est fermé à l'arrêt et interrompt le courant. Un signal LOW désactive le relais (ferme le contact NC1 à nouveau) et permet au courant de circuler de la borne COM1 vers la borne NC1.

Dans notre cas, nous allons utiliser les contacts NO1 et NO2 pour commander les deux lampes.

- **Schéma du montage :**

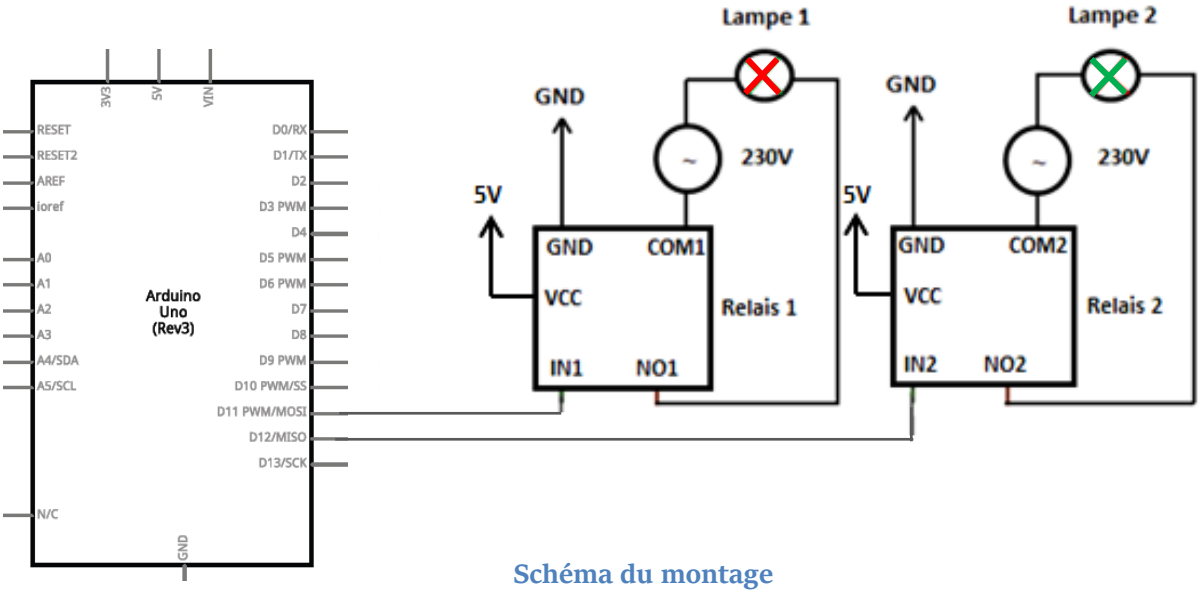
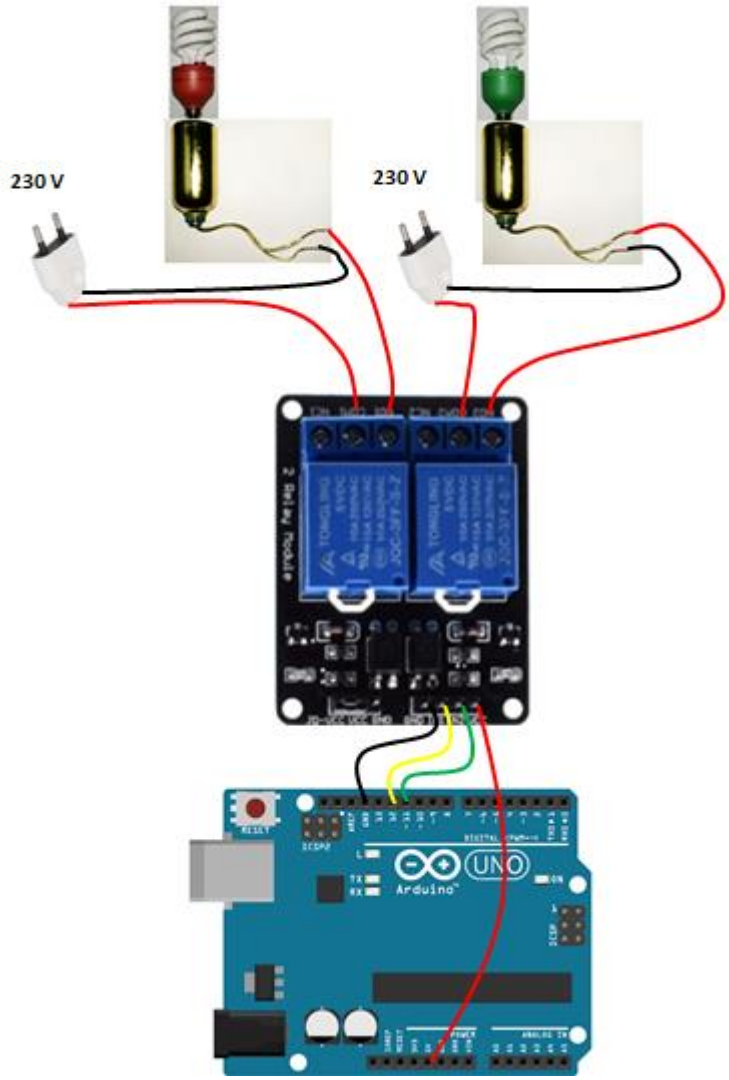


Schéma du montage



Montage de l'application

- **Code du sketch :**

```
int BrocheRel1 = 11; //Entrée IN1 du module lié la sortie numérique 11
int BrocheRel2 = 12; //Entrée IN1 du module lié la sortie numérique 12
unsigned  int Tempo = 5000;     //Déclaration et initialisation d'une temporisation de 5000 ms

void setup() {
pinMode(BrocheRel1, OUTPUT);  // Programmation de la broche 11 comme sortie
pinMode(BrocheRel2, OUTPUT); // Programmation de la broche 12 comme sortie
}

void loop() {
digitalWrite(BrocheRel1, HIGH);  // Mettre la broche 11 sur le niveau haut (5V)
delay(Tempo);                    // Déclencher la temporisation
digitalWrite(BrocheRel1, LOW);   // Mettre la broche 11 sur le niveau bas (0V)
digitalWrite(BrocheRel2, HIGH);  // Mettre la broche 12 sur le niveau haut (5V)
delay(Tempo);                    // Déclencher la temporisation
digitalWrite(BrocheRel2, LOW);   // Mettre la broche 12 sur le niveau bas (0V)
}
```

- **Explications et analyse du sketch :**

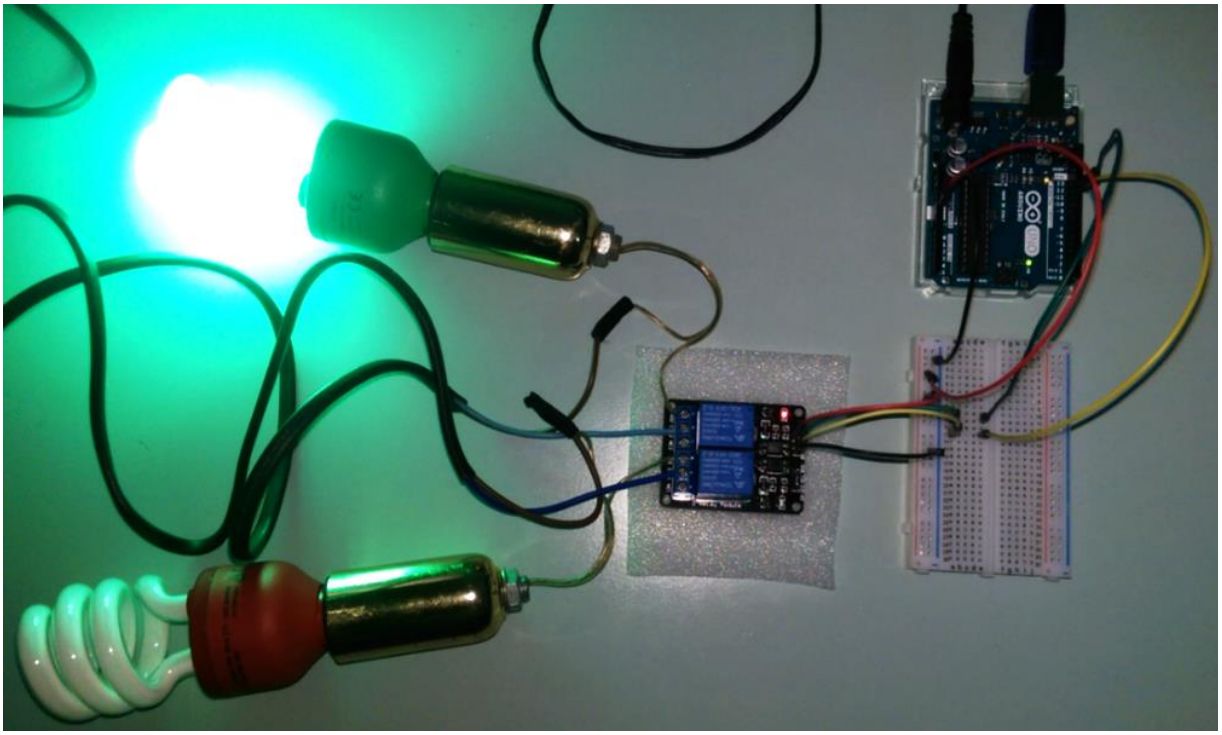
Dans la première partie du sketch, nous avons déclaré et initialiser les broches numériques connectées aux entrées IR1 et IR2 du module à 2 relais, ainsi qu'une variable de temporisation initialisée à la valeur 5000 ms (5 secondes).

Dans la fonction setup, nous avons programmé les deux broches numériques 11 (IR1) et 12 (IR2) en sorties.

Au sein de la fonction loop nous avons défini le code permettant d'assurer le cligonotement de chaque lampe pendant 5 secondes. Voici les différentes étapes du fonctionnement du sketch :

1. Allumer la Lampe rouge (activer le relais 1).

2. Attendre 5 secondes.
3. Eteindre la lampe rouge (désactiver le relais 1) et allumer la lampe verte (activer le relais 2).
4. Attendre 5 secondes.
5. Eteindre la lampe verte (désactiver le relais 2)
6. Revenir à l'étape 1 pour recommencer.



Réalisation du montage

LEÇON 6

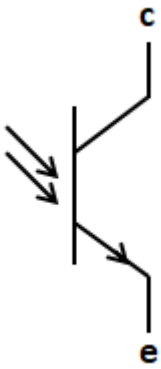
Le Détecteur de flamme

Au sommaire de cette 6ème leçon :

Le détecteur de flamme :

Le capteur de flamme illustré dans la figure ci-dessous est composé d'un phototransistor NPN, il ressemble à une LED, mais il a un boîtier sombre. Ce boîtier bloque la lumière visible et laisse passer l'énergie infrarouge (IR). Les ondes IR ne peuvent pas être vues à l'œil nu, car elles dépassent la partie basse fréquence du spectre de la lumière visible, mais des capteurs comme les phototransistors peuvent détecter leur présence. Les ondes IR émettent normalement à partir de sources de chaleur, comme une flamme ou même une ampoule.

Détecteur de flamme



Symbole électronique d'un phototransistor NPN

Le phototransistor fonctionne comme un transistor standard, mais au lieu d'être activé et désactivé avec une tension de polarisation typique et une source de courant à sa borne de base, la base est régulée par l'effet photoélectrique.

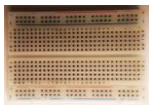
Application : Montage de signalisation d'une flamme

Dans cette application, nous allons réaliser un montage permettant de signaler l'existence d'une flamme avec les spécifications fonctionnelles suivantes :

- En absence de flamme une LED verte s'allume.
- La présence d'une flamme déclenche une sonnette d'alarme d'un buzzer actif et une LED rouge clignote tant que la flamme existe.

Matériels et composants nécessaires :

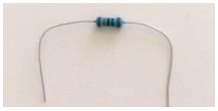

Aduino Uno

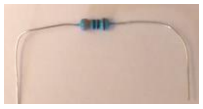

Breadboard


Capteur de flamme


Fils de connexion


2* LED


Résistance 10 kΩ


2 * Résistance 330 Ω


Buzzer actif

Comme vous voyez, nous allons servir d'un buzzer piézoélectrique actif pour émettre un son en cas de présence d'une flamme.

En tant que buzzer électronique à structure intégrée, les buzzers alimentés en courant continu sont largement utilisés dans les ordinateurs, imprimantes, photocopieurs, alarmes, jouets électroniques, dispositifs électroniques automobiles… etc. Les buzzers peuvent être catégorisés comme actifs et passifs (figures suivantes).

Buzzer actif


Buzzer passif

Un buzzer actif a une source oscillante intégrée, de sorte qu'il fera des sons lorsqu'il est alimenté. Mais un buzzer passif n'a pas cette source, donc il ne va pas pépier si des signaux DC sont utilisés; à la place, vous devez utiliser des ondes carrées dont la fréquence est comprise entre 2KHz et 5KHz pour le piloter.

- **Schéma du montage :**

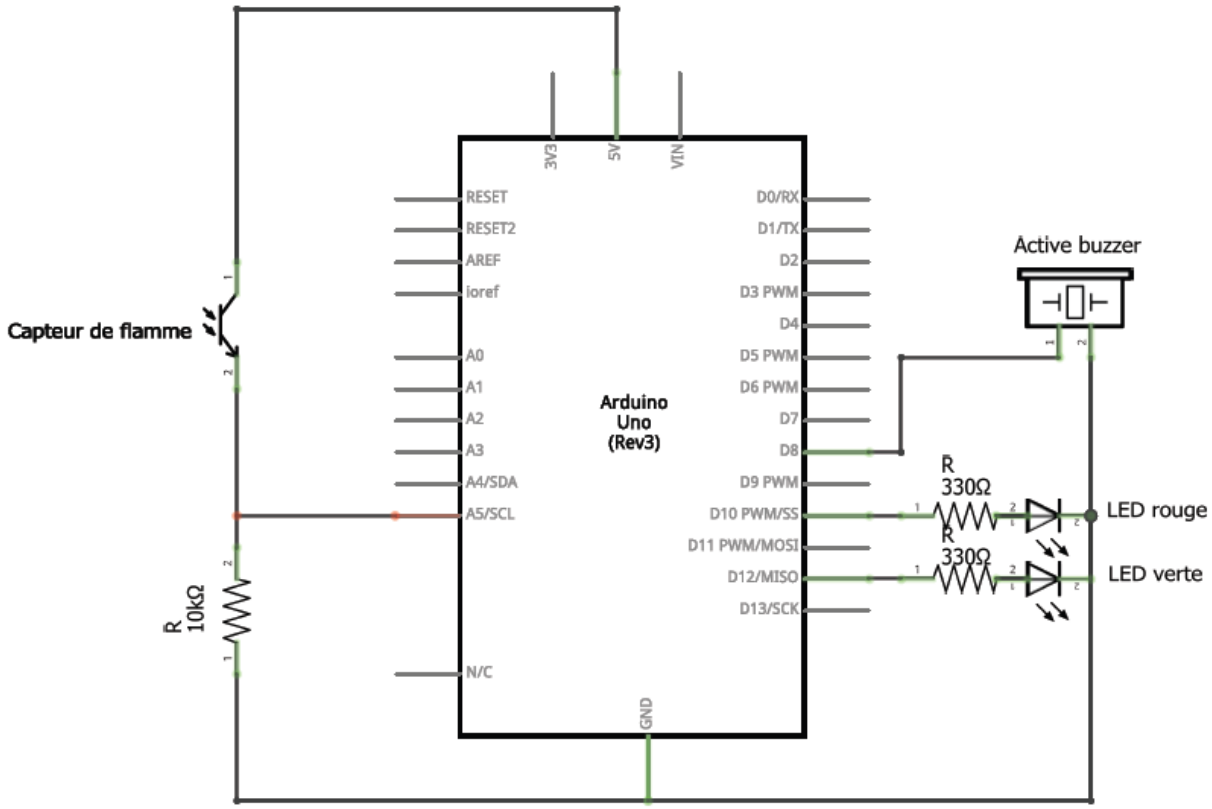
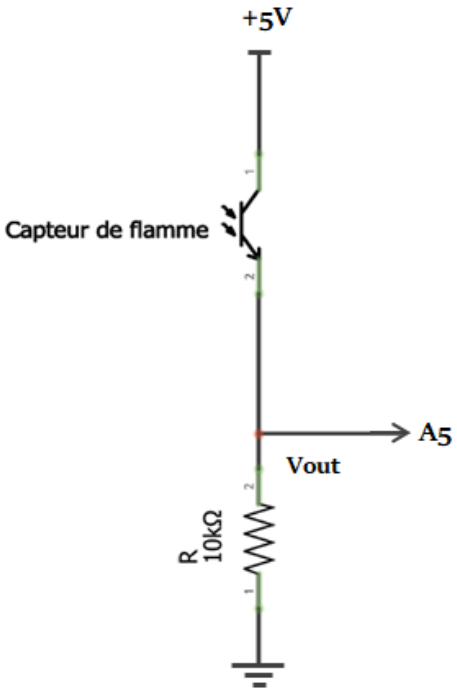


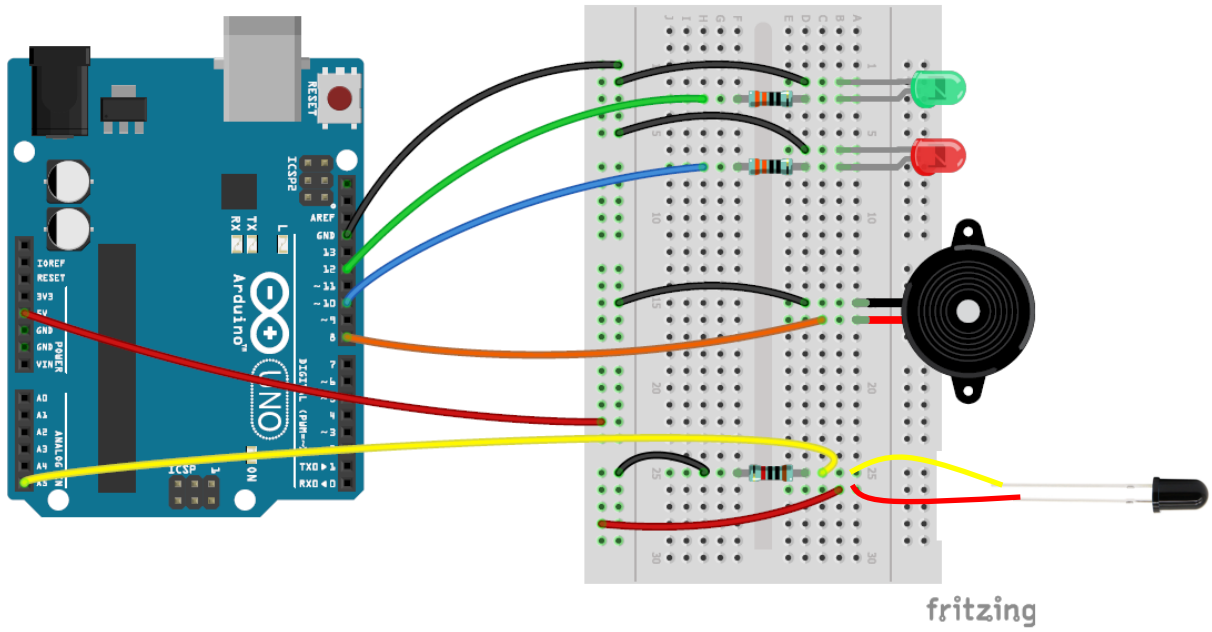
Schéma du montage

Tension à l'entrée analogique 5 :



Tension de sortie du détecteur de flamme

Sans source IR, le phototransistor sera bloqué et VA5 sera 0V. Lorsqu'une source IR est introduite dans le circuit, le phototransistor devient passant permettant au courant de circuler de Vcc (5v) à travers son émetteur et la résistance de 10 kΩ. Ainsi, la tension vue à l'entrée analogique 5 sera fonction de l'intensité de la source IR.



Réalisation du montage avec fritzing

- **Code du sketch :**

```
int EntreeAnalog = 5;   // Entrée analogique 5
int val = 0;             // Variable pour stocker la valeur de l'entrée analogique 5
int Buzzer=8;            //Sortie liée au buzzer
int LedRouge=11;         //Sortie liée à la LED rouge
int LedVerte=12;         //Sortie liée à la LED verte
int tempo=300;           //Temporisation de 300 ms

void setup() {
 pinMode(Buzzer,OUTPUT);       //Programmation de la broche 8 comme sortie
 pinMode(LedRouge,OUTPUT);     //Programmation de la broche 11 comme sortie
 pinMode(LedVerte,OUTPUT);     //Programmation de la broche 12 comme sortie
}
```

```
void loop() {
 val = analogRead(EntreeAnalog);   // Lire la valeur à l'entrée analogique 5
 while(val>=60){    //Tant que la valeur de l'entrée analogique 5 est supérieur ou égale à 60
   digitalWrite(Buzzer, HIGH);       //Activer l'alarme du buzzer
   digitalWrite(LedVerte, LOW);    //Eteindre la LED verte
   blinkLedRouge();      // Appelle de la fonction qui fait clignoter la LED rouge
   val = analogRead(EntreeAnalog);     //Actualiser la variable 'val'
 }
 digitalWrite(Buzzer, LOW);         //Désactiver l'alarme du buzzer
 digitalWrite(LedVerte, HIGH);     //Allumer la LED verte
 }
 void blinkLedRouge(){      //Définition de la fonction du clignotement de la LED rouge
   digitalWrite(LedRouge, HIGH);
   delay(tempo);
   digitalWrite(LedRouge, LOW);
   delay(tempo);
 }
```

- **Explication et analyse du sketch :**

Dans la première partie du sketch, nous avons déclaré et initialisé les différentes variables, ainsi que dans la fonction setup, nous avons programmé les trois broches numériques utilisées en sorties : La broche 8 liée au buzzer, 11 liée à la LED rouge et 12 liée à la LED verte.

Dans la fonction loop, nous avons défini le code permettant de réagir en cas de présence d'une flamme.

```
while(val>=60){    //Tant que la valeur de l'entrée analogique 5 est supérieur ou égale à 60
   digitalWrite(Buzzer, HIGH);       //Activer l'alarme du buzzer
   digitalWrite(LedVerte, LOW);    //Eteindre la LED verte
   blinkLedRouge();      // Appelle de la fonction qui fait clignoter la LED rouge
```

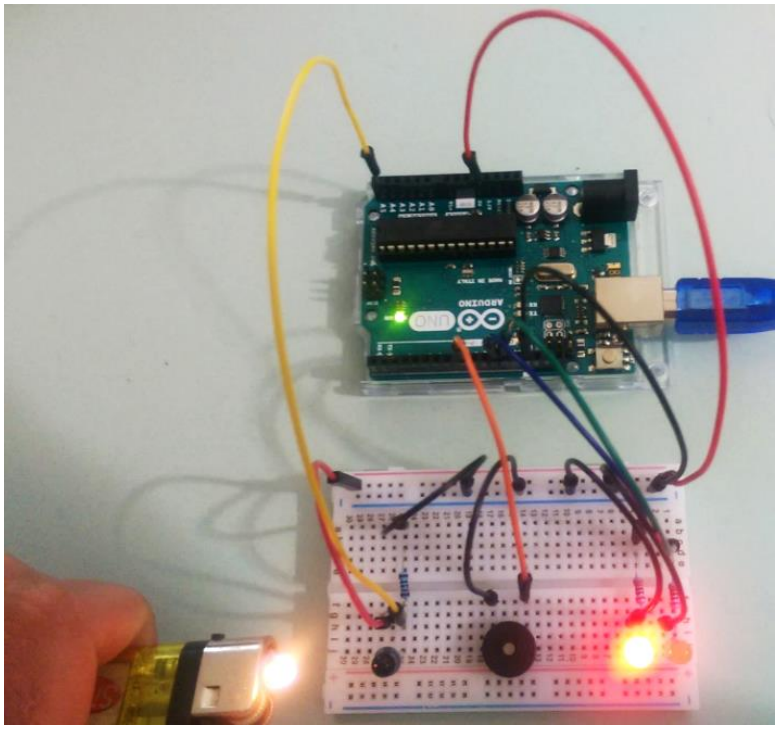
```
  val = analogRead(EntreeAnalog);     //Actualiser la variable 'val'
}
```

On lit d'abord la valeur de conversion de la tension de sortie du détecteur de flamme à l'entrée analogique 5, et tant que ce nombre est supérieur ou égal à 60 (présence de flamme), le buzzer s'active pour émettre une alarme, la LED rouge s'éteint et la LED rouge commence à clignoter (appelle de la fonction du clignotement qui est défini dehors la fonction loop), puis on actualise l'état de la variable 'val' pour un nouveau teste au sein de la boucle while.

Si la condition sur la boucle while est non valide (c'est-à-dire val<60), le buzzer doit être désactivé, la LED rouge s'arrête de clignoter et la LED verte s'allume pour indiquer l'absence d'une flamme :

```
digitalWrite(Buzzer, LOW);        //Désactiver l'alarme du buzzer
digitalWrite(LedVerte, HIGH);     //Allumer la LED verte
```

Puis le cycle recommence dès le début de la fonction loop.



Réalisation du montage

LEÇON 7

L’émetteur et le récepteur infrarouge

Au sommaire de cette 7ème leçon :

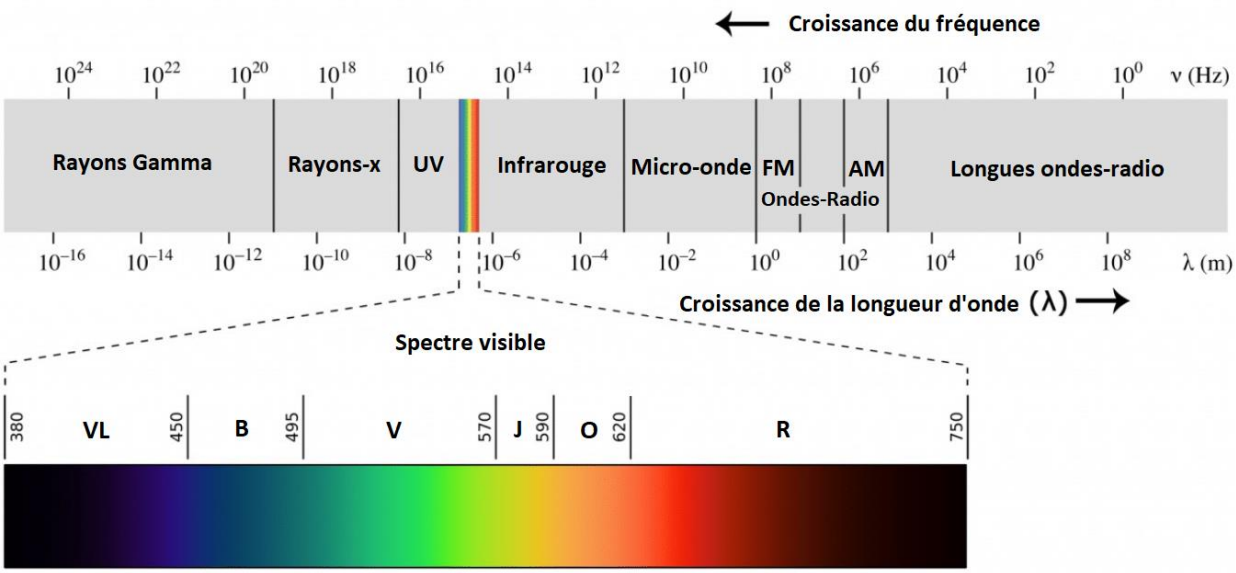
Communication infrarouge (IR) :

La communication infrarouge (IR) est une technologie sans fil largement utilisée et facile à mettre en œuvre qui a de nombreuses applications utiles. Les exemples les plus frappants dans la vie de tous les jours sont les télécommandes TV / vidéo, les détecteurs de mouvement et les thermomètres infrarouges.

Avec un émetteur et un récepteur IR simples, nous pouvons réaliser des projets Arduino intéressants qui utilisent la communication (robots télécommandés, des capteurs de distance, des télécommandes d'appareils photo reflex numériques, des télécommandes TV ...etc).

- **Qu'est-ce que l'infrarouge ?**

Le rayonnement infrarouge (rayonnement électromagnétique), est une forme de lumière semblable à la lumière que nous voyons autour de nous, la seule différence entre la lumière IR et la lumière visible est la fréquence et la longueur d'onde. Le rayonnement infrarouge a une longueur d'onde supérieure à celle du spectre visible de sorte que les humains ne peuvent pas le voir.



Spectre des rayonnements électromagnétiques en fonction de leur longueur d'onde et leur fréquence.

Vu que l'infrarouge est un type de lumière, la communication IR nécessite une ligne de visée directe de l'émetteur au récepteur IR. Un émetteur IR ne peut pas transmettre une donnée au récepteur à travers les murs ou d'autres matériaux comme dans le cas de la technologie WiFi ou Bluetooth.

- **Principe de fonctionnement d'une télécommande et récepteur infrarouge :**

Un système de communication infrarouge typique nécessite un émetteur IR et un récepteur IR. L'émetteur ressemble à une LED standard, sauf qu'il produit de la lumière dans le spectre IR au lieu du spectre visible. Si vous regardez à l'entête d'une télécommande TV, vous verrez la LED de l'émetteur IR (Figure à droite).

LED IR à l'entête d'une télécommande IR

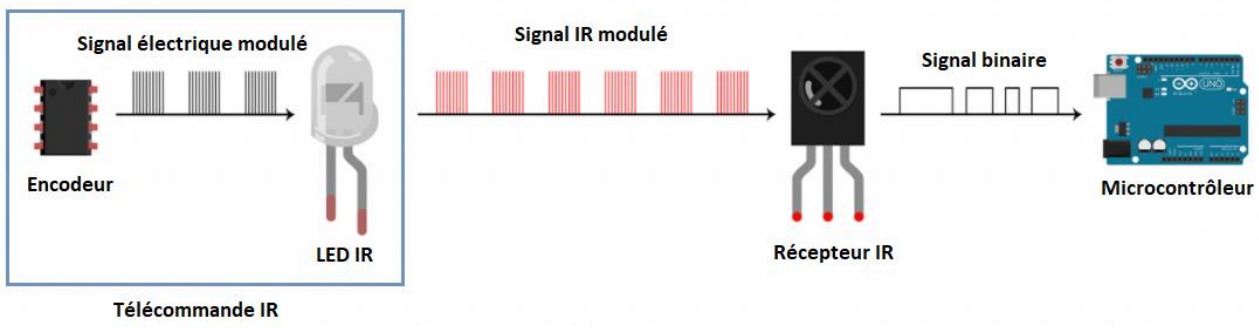
Récepteur IR

Le récepteur IR est une photodiode et un préamplificateur qui convertit la lumière infrarouge en un signal électrique. Les diodes réceptrices IR ressemblent généralement à celle représentée dans la figure à gauche

Modulation d'un signal IR :

La lumière IR est émise par le soleil, les ampoules et tout ce qui produit de la chaleur. Cela signifie qu'il y a beaucoup de bruit de lumière IR autour de nous. Pour éviter que ce bruit n'interfère avec le signal IR, une technique de modulation du signal est utilisée.

En modulation du signal IR (figure ci-dessous), un encodeur sur la télécommande IR convertit un signal binaire en un signal électrique modulé. Ce signal électrique est envoyé à la LED d'émission (LED IR). La LED d'émission convertit le signal électrique modulé en un signal de lumière IR modulé. Le récepteur IR démodule alors le signal lumineux IR et le convertit en un signal binaire avant de transmettre l'information à un microcontrôleur.



Modulation d'un signal IR transmis par une télécommande à un μc via un récepteur IR

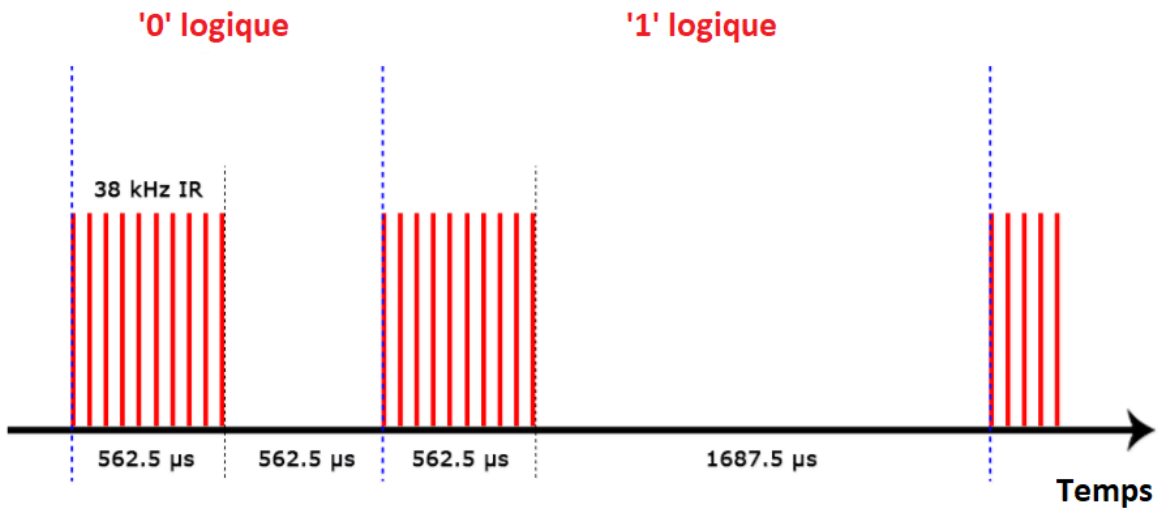
Le signal IR modulé est une série d'impulsions lumineuses infrarouges allumées et éteintes à une fréquence élevée connue sous le nom de fréquence porteuse. La fréquence porteuse utilisée par la plupart des émetteurs est de 38 kHz, car elle est rare dans la nature et peut donc être distinguée du bruit ambiant. De cette façon, le récepteur IR saura que le signal de 38 kHz a été envoyé par l'émetteur et qu'il n'a pas été capté dans l'environnement.

La diode du récepteur détecte toutes les fréquences de la lumière IR, mais elle possède un filtre passe-bande et ne laisse passer que l'infrarouge à 38 kHz. Elle amplifie ensuite le signal modulé avec un préamplificateur et le convertit en un signal binaire avant de l'envoyer à un microcontrôleur.

Protocoles de transmission IR :

Le modèle dans lequel le signal IR modulé est converti en binaire est défini par un protocole de transmission. Il existe de nombreux protocoles de transmission IR : Sony, Matsushita, NEC et RC5 sont quelques-uns des protocoles les plus courants.

Le protocole NEC est également le type le plus utilisé dans les projets Arduino, je vais donc l'utiliser comme exemple pour vous montrer comment le récepteur convertit le signal IR modulé en un signal binaire.



Principe de transmission d'un signal IR modulé en binaire avec le protocole NEC

Le '**1**' logique s'envoie avec une impulsion HIGH de 562,5 µs et de fréquence 382 kHz suivie d'une longue impulsion LOW de 1687,5 µs. Le '**0**' logique est transmis avec une impulsion HIGH de 562,5 µs suivie d'une impulsion LOW de 562,5 µs.

C'est de cette manière que le protocole NEC code et décode les données binaires en un signal modulé. Les autres protocoles ne diffèrent que par la durée des impulsions individuelles HAUTE et BASSE.

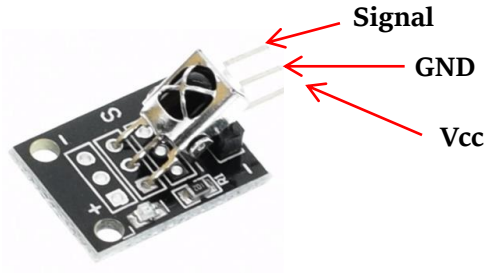
Les codes infrarouges :

Chaque fois que vous appuyez sur un bouton de la télécommande, un code hexadécimal unique est généré. C'est l'information qui est modulée et envoyée par IR au récepteur. Pour déchiffrer quelle touche est enfoncée, le microcontrôleur récepteur doit savoir quel code correspond à chaque touche de la télécommande.

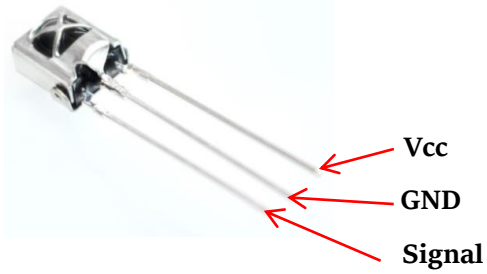
Différentes télécommandes peuvent envoyer différents codes pour une même touche, vous devrez donc déterminer le code généré pour chaque touche de votre télécommande. Si vous pouvez trouver la fiche technique, les codes de clé IR doivent être listés. Sinon, il y a un simple sketch Arduino qui va lire la plupart des télécommandes populaires et imprimer les codes hexadécimaux sur le moniteur série lorsque vous appuyez sur une touche. Nous allons voir comment le faire tout de suite, mais d'abord, nous devons savoir comment connecter un récepteur IR à la carte Uno.

Brochage du récepteur IR :

Il existe plusieurs types de récepteurs IR, certains sont autonomes et d'autres sont montés sur une carte de dérivation (Figures ci-dessous). Vérifiez la fiche technique de votre récepteur IR particulier, car les broches peuvent être disposées différemment du récepteur IR HX1838 que nous allons utiliser. Cependant, tous les récepteurs IR auront trois broches: signal, masse et Vcc.



Récepteur IR monté sur une carte



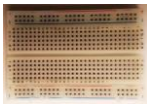
Récepteur IR autonome

- **Afficher les codes des touches de votre télécommande sur le moniteur série :**

Afin de trouver les codes des touches de votre télécommande, nous allons connecter le récepteur IR avec l'Arduino et charger un sketch permettant d'imprimer le code hexadécimal de chaque touche appuyée sur la télécommande sur le moniteur série de l'IDE Arduino.

Matériels et composants nécessaires :

Aduino Uno



Breadboard

Fils de connexion

Récepteur IR HX1838

Schéma du montage :

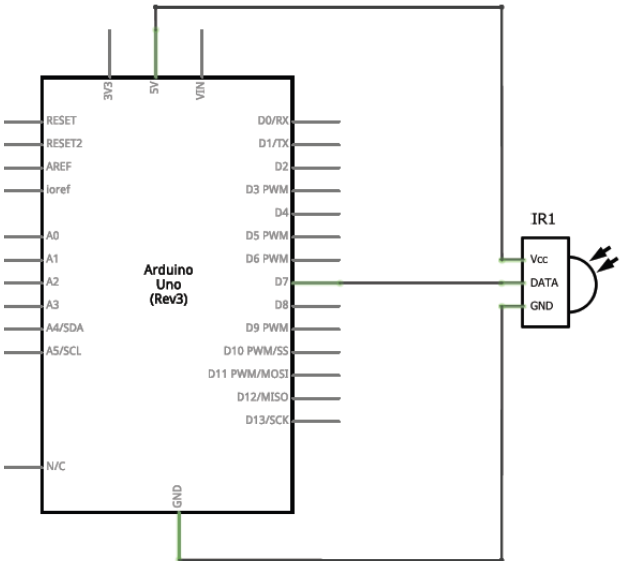
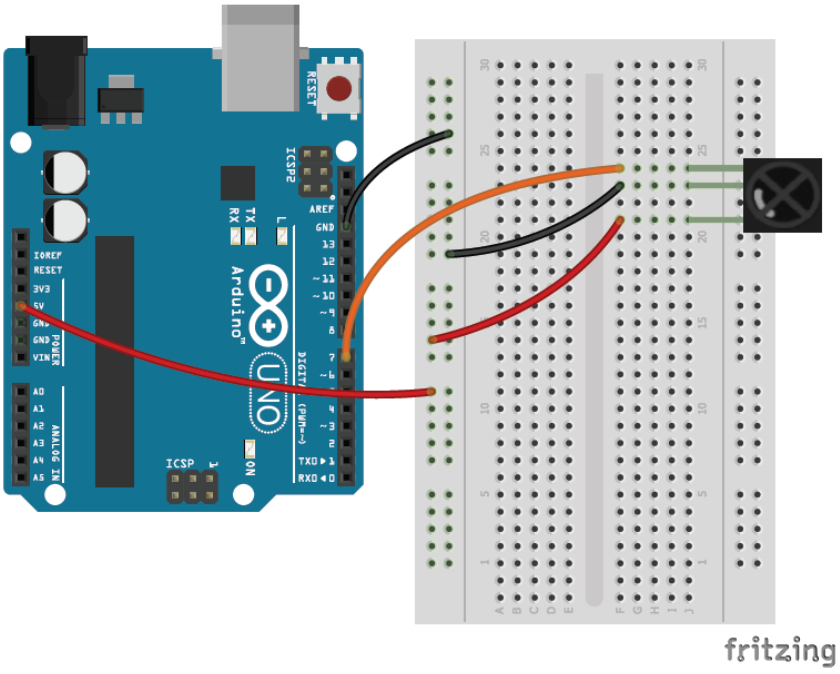


Schéma du montage



Réalisation du montage avec fritzing

Code du sketch :

Dans ce sketch nous allons utiliser la bibliothèque 'IRremote'. Pour installer cette bibliothèque, Rendez-vous sur la barre de menu de l'IDE Arduino, puis accédez à **Croquis** > **Inclure une bibliothèque** > **Gérer les bibliothèques** et cherchez "IRremote". Cliquez sur la bibliothèque, puis cliquez sur **installer**.

```
#include <IRremote.h>
const int Broche_Recep = 7; //Broche numérique liée au récepteur IR
IRrecv recep(Broche_Recep); //Instanciation de l'objet 'recep'
decode_results result;  //Instanciation de l'objet 'result'

void setup(){
 Serial.begin(9600);  //définition de le vitesse de transmission (baud)
 recep.enableIRIn();  //Démarrer le processus de réception
 recep.blink13(true);  //Activer le clignotement de la LED 13 en cours de réception
}

void loop(){
 if (recep.decode(&result)){  //Si un code est reçu
      Serial.println(result.value, HEX);  //Afficher le code reçu en hexadécimal
      recep.resume();  //Réinitialiser le processus de réception
 }
}
```

Explication et analyse du sketch :

Pour une communication IR utilisant la bibliothèque IRremote, nous avons d'abord créée un objet appelé 'recep' et spécifier le numéro de broche où le récepteur IR est connecté (ligne 2 et 3). Cet objet prendra en charge le protocole et le traitement des informations provenant du récepteur.

L'étape suivante consiste à créer un objet appelé 'result', à partir de la classe decode_results, qui sera utilisée par l'objet 'recep' pour partager les informations décodées avec notre application (ligne 4).

Dans le bloc de la fonction setup, nous avons défini la vitesse de transmission du moniteur série. Ensuite, nous avons démarré le récepteur IR en appelant la méthode enableIRIn().

La méthode blink13(true) permet de faire clignoter la LED 13 de l'Arduino à chaque fois que le récepteur reçoit un signal de la télécommande, ce qui est utile pour donner une rétroaction visuelle.

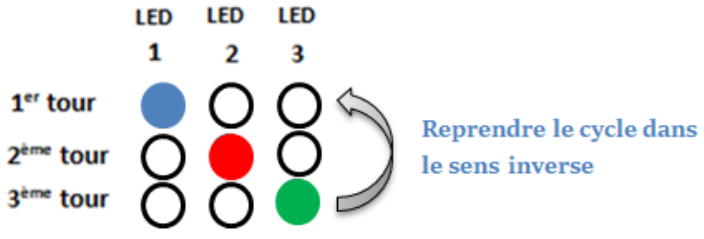
Dans le bloc de la fonction loop, la méthode decode(&result) retournera vrai si un code est reçu et que le programme exécutera le code dans l'instruction if. Le code reçu est stocké dans result.value. On affiche alors ce code en hexadécimal sur le moniteur série.

À la fin du bloc de la fonction loop, nous avons appelé la méthode resume() pour réinitialiser le récepteur et le préparer pour recevoir le code suivant.

Application : Commande d'un séquenceur de lumière avec une télécommande et récepteur IR

Dans cette application, nous allons commander un séquenceur de lumière composé de 3 LED (rouge, bleue et noire) à 4 vitesses, avec une télécommande et un récepteur IR, le mode de fonctionnement est comme suite :

- Les LED devront s'allumer conformément au modèle suivant :



Modèle de fonctionnement du séquenceur de lumière

- La touche '>||' commande le démarrage ou bien l'arrêt du séquenceur (touche de marche-arrêt).

- La première vitesse avec laquelle démarre le séquenceur est 1 seconde, ça veut dire que chaque LED doit être allumée pendant une durée de 1 secondes.
- Lorsqu'on appuie une seule fois sur la touche '>>|', la vitesse du séquenceur accélère (basculement à la 2ème vitesse) et devient 500 millisecondes.
- Lorsqu'on appuie une 2ème fois sur la touche '>>|', la vitesse du séquenceur accélère encore plus (basculement à la 3ème vitesse) et devient 250 millisecondes.
- Lorsqu'on appuie une 3ème fois sur la touche '>>|', la vitesse du séquenceur accélère encore plus (basculement à la 4ème vitesse) et devient 100 millisecondes.
- De la même manière, la vitesse ralentit à chaque pression sur la touche '|<<'.

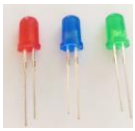
Matériels et composants nécessaires :

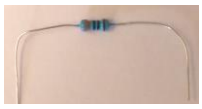

Aduino Uno


Breadboard


Fils de connexion


Récepteur IR HX1838


3 * LED


3 * Résistance 330 Ω

- **Schéma du montage**

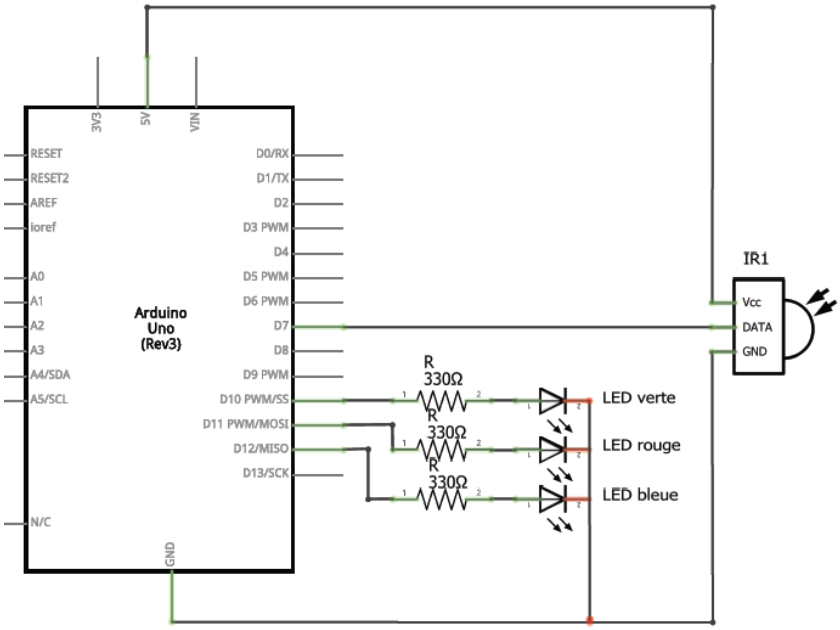
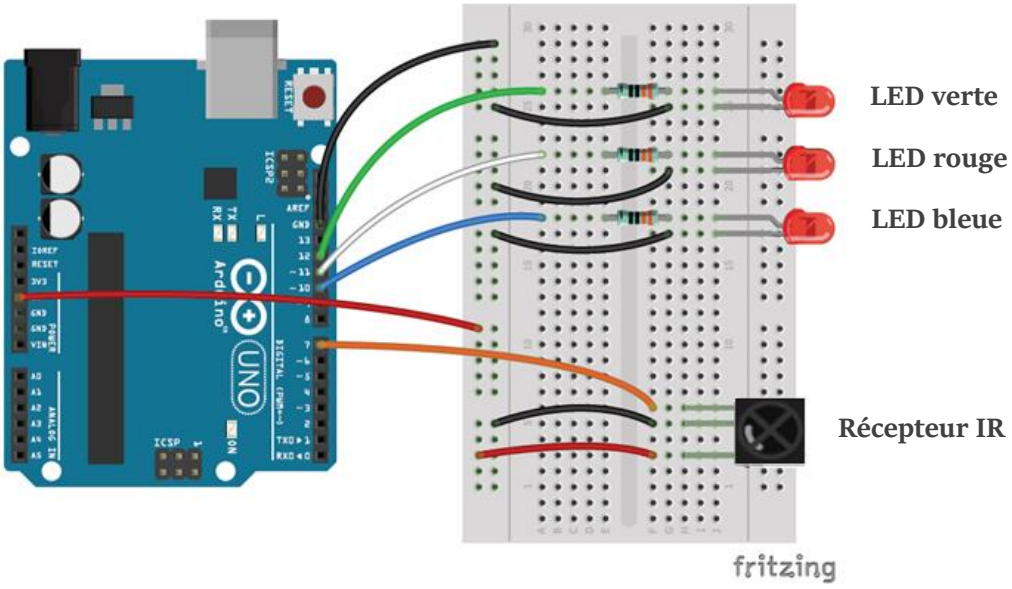


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

```
/*1-Déclaration, initialisation des variables et instanciation des objets */

#include <IRremote.h>
const int RECV_PIN = 7; //Broche numérique liée au récepteur IR
IRrecv irrecv(RECV_PIN); //Instanciation de l'objet 'irrecv'
decode_results results; //Instanciation de l'objet 'results'
int pinray[]={10, 11, 12}; //Tableau des broches LED
int intervalle = 1000;   // Intervalle de temps (1 seconde)
unsigned long prev;    // Variable de temps
int k = HIGH; //variable pour le marche-arrêt du séquenceur
int i=0;
int j=0;
int vitesse=0;
/*2-Fonction setup*/
void setup(){
for(int i=0; i<4; i++)
```

```
pinMode(pinray[i], OUTPUT); //Programmation des broches numériques des LED en sorties
prev = millis();              // Mémoriser le compteur de temps actuel
irrecv.enableIRIn();    //Démarrage du processus de réception
irrecv.blink13(true);  //Activer le clignotement de la LED 13 en cours de réception
}

void loop(){
if(k==LOW){ //si k==LOW le séquenceur démarre
if((millis() - prev) > intervalle ){ //Ecoulement du retard 'intervalle'
prev = millis();
/*3-Programme permettant d'allumer et d'éteindre les LED du séquenceur*/
switch(j){
        case 0:
        digitalWrite(pinray[1], LOW);
        digitalWrite(pinray[0], HIGH);
        break;
        case 1:
        digitalWrite(pinray[0], LOW);
        digitalWrite(pinray[1], HIGH);
        break;
        case 2:
        digitalWrite(pinray[1], LOW);
        digitalWrite(pinray[2], HIGH);
        break;
        case 3:
        digitalWrite(pinray[2], LOW);
        digitalWrite(pinray[1], HIGH);
  }
  j++;
  if(j>3){j=0;}
}
 }
```

```
/*4-Programme permettant de démarrer ou d'arrêter le séquenceur si la touche ">||" est appuyée */
 if (irrecv.decode(&results)){ // Si une touche de la télécommande est appuyée
switch(results.value){
        case 0xFFC23D: // Si la touche ">||" est appuyée
        k=!k;
        if(k==HIGH) //Eteindre toutes les LED avant d'arrêter le séquenceur
        for(int i=0; i<4; i++)
        digitalWrite(pinray[i], LOW);
        break ;
        case 0xFF02FD: // Si la touche ">>|" est appuyée
        /*5-Programme permettant de gérer l'accélération de la vitesse*/
        vitesse++;
        if(vitesse == 1)
        intervalle = 500;
        else if(vitesse == 2)
        intervalle = 250;
        else
        intervalle = 100;
        if(vitesse>3)
        vitesse=3;
        break ;
        case 0xFF22DD: // Si la touche "<<"   est appuyée
        /*6-Programme permettant de gérer le ralentissement de la vitesse*/
        vitesse--;
        if(vitesse == 3)
        intervalle = 100;
        else if(vitesse == 2)
        intervalle = 250;
        else if(vitesse == 1)
        intervalle = 500;
        else
```

```
      intervalle = 1000;
      if(vitesse<0)
      vitesse=0;
      break ;
      }
     irrecv.resume(); //Réinitialiser le processus de réception
  }
 }
}
```

- **Explication et analyse du sketch :**

Le code du sketch peut être divisé en 6 parties :

1- La première partie est consacrée à la déclaration et l'initialisation des variables ainsi que instanciation des objets.

2- Dans la deuxième partie et au sein de la fonction setup, nous avons programmé les 3 broches numériques (pour les LED) en sorties, aussi nous avons démarré le processus de réception, initialisé la variable du temps 'prev' avec la valeur renvoyé par la fonction millis() et activé le clignotement de la LED 13 en cours de la réception .

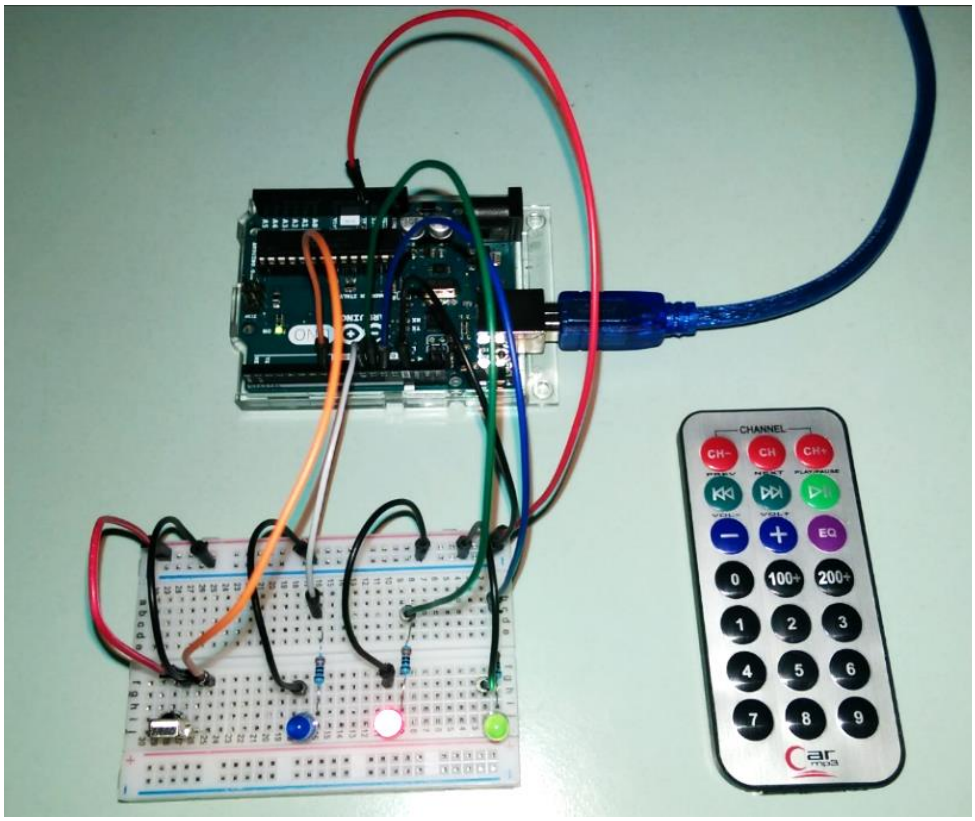
3- Dans cette partie nous avons défini le code du programme permettant d'allumer et d'éteindre les LED du séquenceur, le fonctionnement est géré avec une switch de la variable 'j' après l'écoulement du retard 'intervalle' qui est géré par l'instruction :

```
if((millis() - prev) > interval )
```

4- Dans cette 4ème partie nous avons défini le code du programme permettant de démarrer ou d'arrêter le séquenceur si la touche ">||" est appuyée, le processus est lié avec l'état de la variable 'k', si k=LOW le séquenceur fonctionne sinon (k=HIGH) le séquenceur s'arrête en mettant toutes les broches des LED à l'état LOW.

5- Dans cette partie nous avons défini le code permettant de gérer l'accélération de la vitesse du séquenceur si la touche '>>|' est appuyée, si le code stocké dans results.value correspond à celui de la touche '>>|', le contenu de la variable du retard 'intervalle' sera changé en fonction du nombre de pressions sur la touche '>>|'.

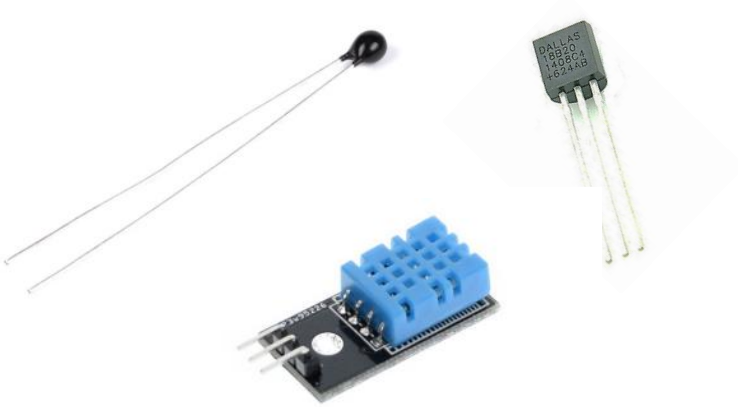
6- Dans la 6ème et dernière partie, nous avons défini le code traitant le ralentissement de vitesse du séquenceur si la touche '|<<' est appuyée. si le code stocké dans results.value correspond à celui de la touche '|<<', le contenu de la variable 'intervalle' sera changé en fonction du nombre de pressions sur la touche '|<<'.



Réalisation du montage

LEÇON 8

Les capteurs de température



Au sommaire de cette 8ème leçon :

La température est un paramètre climatique qui joue un rôle prépondérant dans la majorité des phénomènes physiques. Pour la mesurer, on utilise des capteurs de température analogiques ou bien des capteurs numériques.

Les capteurs analogiques produisent un signal de sortie analogique continu qui est proportionnel à la grandeur physique mesurée (mesurande).

Les capteurs numériques intègrent tout le nécessaire requis pour faire la mesure : capteur analogique, convertisseur analogique/numérique, électronique de communication et alimentation. Le choix d'un capteur de température dépend de l'application et des caractéristiques du capteur lui-même (plage de fonctionnement, sensibilité, linéarité, temps de réponse, robustesse, etc.).

La thermistance :

Une thermistance est un type de résistance dont la résistance varie en fonction de la température. On distingue deux types fondamentaux de thermistances : Les CTN et les CTP. Dans les thermistances CTN (Coefficient de Température Négatif), la résistance diminue à mesure que la température augmente, tandis qu'avec les thermistances CTP, la résistance augmente lorsque la température augmente.

Les thermistances sont largement utilisées comme capteurs de température et limiteurs de courant d'appel (Type CTN) et pour la protection contre les surintensités avec réarmement automatique et éléments chauffants autorégulants (type CTP).

Les thermistances CTN sont les plus courantes, et c'est le type que nous utiliserons dans cette leçon pour mesurer la température. Une CTN (figure à droite) est fabriquée à partir d'un matériau semi-conducteur (tel qu'un oxyde métallique ou la céramique) qui a été chauffé et comprimé pour former un matériau conducteur sensible à la température.

Thermistance CTN utilisée : MF52 10KΩ

Les thermistances CTN sont des composants simples, peu coûteux et précis qui facilitent l'obtention des données de température. Ce sont des capteurs analogiques, donc le code du programme sera relativement simple par rapport aux capteurs de température numériques qui nécessitent des bibliothèques spéciales et du code assez compliqué.

Les thermistances ont l'avantage d'une très grande sensibilité aux changements de température, mais leur inconvénient majeur c'est qu'elles possèdent une caractéristique R=f(T) non linéaire.

La forme la plus couramment utilisée pour mesurer la température avec une thermistance CTN est l'équation de Steinhart : $\frac{1}{T} = A + B\ln(R) + C(\ln(R))^3$

Avec :

T : Sa température (en kelvins).

R : Sa résistance électrique (en ohms).

A, **B** et **C** : Sont les coefficients de Steinhart qui caractérisent chaque thermistance.

L'équation contient aussi, en théorie, un terme en $(\ln(R))^2$, généralement négligeable devant les autres coefficients. C'est pourquoi il n'est pas considéré ici. (En sa présence il y aurait alors 4 coefficients.)

Les coefficients de Steinhart sont parfois publiés par les fabricants. Si ce n'est pas le cas, pour les trouver il faut résoudre un système à 3 équations et 3 inconnues en exploitant les mesures de R en fonction de T fournies par le constructeur.

Exemple : **Thermistance MF52 10K B3950**

Résistance (Ω)	T (°C)
5	**25071**
25	**10000**
50	**3572**

Trois mesures de R =f(T) extraites de la fiche technique du constructeur

On se basant sur l'équation de Steinhart, on peut maintenant utiliser les trois paires de valeurs de mesure (R=f(T)) pour former un système de trois équations à trois inconnues (Coefficients A, B et C). Nous les écrivons sous forme matricielle ci-dessous :

$$\begin{bmatrix} 1 & 25071 & (\ln(25071))^3 \\ 1 & 10000 & (\ln(10000))^3 \\ 1 & 3572 & (\ln(23572))^3 \end{bmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{bmatrix} \frac{1}{5+273.15} \\ \frac{1}{25+273.15} \\ \frac{1}{50+273.15} \end{bmatrix}$$

Vous pouvez aussi utiliser des calculateurs de coefficients d'une CTN disponibles sur internet où il suffit d'entrer les valeurs de mesures de R et T et d'obtenir directement les valeurs de A, B et C. Les valeurs de A, B et C ainsi obtenues après résolution du système d'équation sont :

A=1.285020346^{e}-3 ; B=2.082927340^{e}-4 ; C=1.926864414^{e}-7

Maintenant il reste à implanter un programme pour mesurer R instantanément et obtenir la valeur de température. Pour ce faire, il faut monter la CTN en série avec une résistance de 10kΩ dans un montage diviseur de tension. Les formules pour mesurer R et T sont :

$$\boldsymbol{R(\Omega) = R1\ \frac{1023}{V0-1}} \quad ; \quad \boldsymbol{T(°K) = \frac{1}{A + B\ln(R) + C\,(\ln(R))^3}}$$

Avec :

R : Valeur de la résistance de la CTN (en Ω).

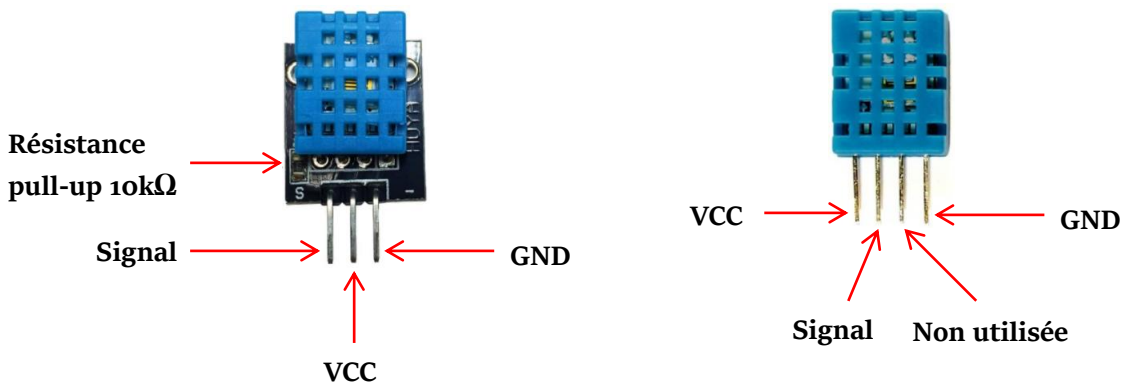
R1 : Résistance de 10kΩ.

Vo : Tension aux bornes de la CTN mesurée à l'entrée analogique de l'Arduino.

Le capteur DHT11 :

Le DHT11 est un capteur numérique qui mesure la température et l'humidité relative. Sa plage de mesure est comprise entre 0° et 50°C (avec une précision de 2%) pour la température et entre 20% et 90% (avec une précision de 5%) pour l'humidité. Le DHT11 utilise un seul fil de signal pour transmettre des données à un microcontrôleur. L'alimentation provient de fils séparés de 5V et de terre. Une résistance pull-up de 10KΩ est nécessaire entre la ligne de signal et la ligne 5V pour s'assurer que le niveau du signal reste élevé par défaut (voir la fiche technique pour plus d'informations).

Il y a deux versions différentes du DHT11 que vous pourriez rencontrer (figures ci-dessous) :



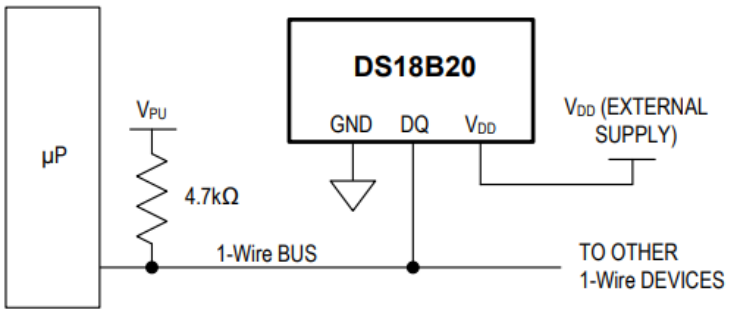
Brochage des deux version du capteur

Un type a quatre broches, et l'autre type a trois broches et est monté sur un petit circuit imprimé. La version montée sur un circuit imprimé est agréable car elle comprend une résistance de tirage (pull-up) de 10KΩ montée en surface pour la ligne du signal.

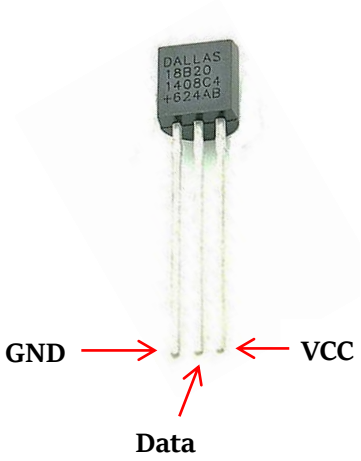
Il existe une bibliothèque « DHT » que nous allons utiliser par la suite et qui regroupe toutes les méthodes nécessaires pour obtenir les lectures de mesure d'humidité et de température du capteur.

Le capteur DS18B20 :

Le capteur DS18B20 est un capteur de température numérique qui communique via un bus 1-Wire et possède une résolution numérique de 12 bits (programmable) avec une plage de mesure de -55°C à +125°C. La précision analogique du capteur est de 0,5°C entre -10°C et +85°C, ce qui rend ce capteur très intéressant pour une utilisation normale.



Câblage du capteur DS18B20 sur un bus 1-wire selon la fiche du constructeur



Capteur DS18B20 en boitier TO-92 avec son brochage

Le DS18B20 communique sur un bus 1-Wire qui, par définition, ne nécessite qu'une seule ligne de données (data), une ligne d'alimentation (5V) et la terre pour une communication avec un microcontrôleur.

Chaque DS18B20 possède un code série unique de 64 bits qui permet à plusieurs DS18B20 de fonctionner sur le même bus 1-Wire. Ainsi, il est simple d'utiliser un seul microcontrôleur pour contrôler de nombreux DS18B20 répartis sur une grande surface.

Il existe aussi pour ce capteur une bibliothèque « OneWire » contenant des méthodes qui permettent d'extraire facilement la mesure de la température du capteur et de faire gérer aussi plusieurs capteurs sur le même bus 1-wire.

Affichage de la température mesurée sur le moniteur série :

Nous allons voir dans cette partie, comment mesurer la température avec les 3 capteurs (cités précédemment) à la fois, et afficher les résultats de mesure sur le moniteur série de l'IDE Arduino.

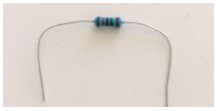
Matériels et composants nécessaires :

Aduino Uno

Breadboard

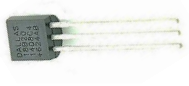
Fils de connexion

Thermistance MF52 10KΩ



Résistance 10 kΩ

Module du capteur DHT11



Capteur DS18B20

Résistance 4.7 kΩ

- **Schéma du montage :**

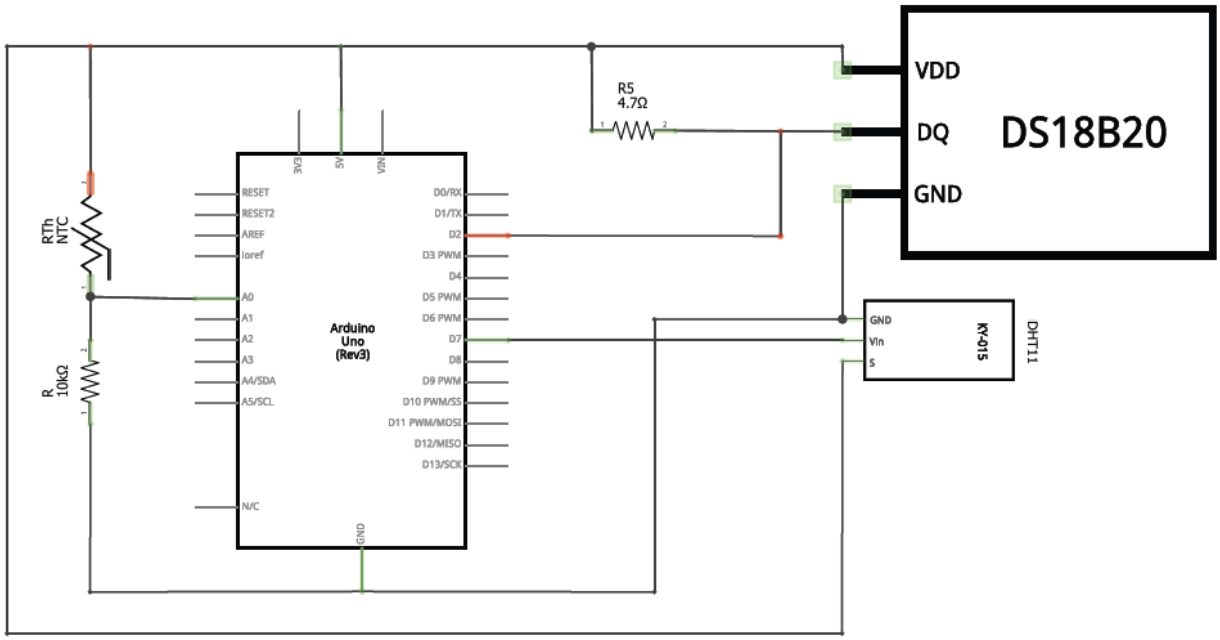
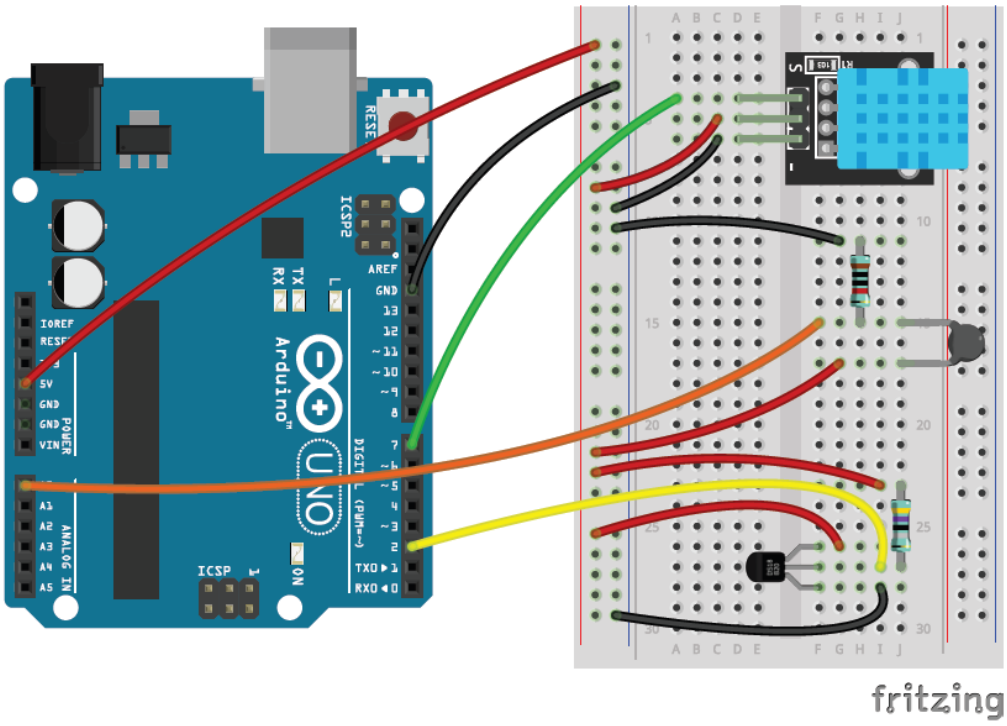


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

Dans ce sketch nous allons utiliser 3 bibliothèque : 'OneWire', 'DallasTemperature' et 'dht'. Pour les installer, rendez-vous sur la barre de menu de l'IDE Arduino, puis accédez à **Croquis** > **Inclure une bibliothèque** > **Gérer les bibliothèques** et cherchez le nom de la bibliothèque, puis cliquez sur la bibliothèque, et cliquez sur **installer**.

```
/*1-Incorpoation des bibliothèques, définition de constantes, déclaration des variables et instanciation des objets*/
#include <OneWire.h>
#include <DallasTemperature.h>
#include <dht.h>
#define ONE_WIRE_BUS 2 // Broche de données du bus oneWire connécté sur la broche 2
#define DHT11_PIN 7 //Dht11 sur la sortie numérique 7
int BrocheTh = 0; //Tension de sortie de la thermistance sur l'entrée analogique 0
int VTh; //Tension de sortie de la thermistance
```

```
float R1 = 10000; //Résistance 10kOhms
float RTh, Tk, Tc; //Résistance de la thermistance, température en °Kelvin et température en
//°Celcius
float a = 1.129876184e-3 , b = 2.348827851e-4 , c = 0.7780438126e-7;//Coefficients de
//Steinhart
dht DHT;
OneWire oneWire_DS18B20(ONE_WIRE_BUS);// Créer une instance d'objet oneWire pour
//communiquer avec tous les périphériques OneWire
DallasTemperature sensors(&oneWire_DS18B20);//Transmettre notre référence oneWire au
//capteur de température Dallas

void setup()
{
 /*2-Vitesse de transmission du moniteur série et*/
 Serial.begin(9600);
 // Start up the library
 sensors.begin(); //Initialiser le capteur DS18B20
}

void loop(){
 /*3-Mesure de température avec la thermistance NTC MF52 10K*/
 VTh = analogRead(BrocheTh);
 RTh = R1 * (1023.0 / (float)VTh - 1.0);
 Tk = 1.0 / (a + (b*log(RTh)) + (c*pow(log(RTh),3)));
 Tc = Tk - 273.15;
 /*4-Mesure de température avec le capteur DHT11*/
 int resDHT11 = DHT.read11(DHT11_PIN); //Lecture de la température mesurée par DHT11
 /*5-Mesure de température avec le capteur DS18B20*/
 sensors.requestTemperatures(); // Demande de mesure de température
 /*6-Affichage des mesures de température */
 Serial.print("temperature DS18b20: ");
```

```
 Serial.print(sensors.getTempCByIndex(0)); //Afficher la température mesurée par DS18B20,
//vous pouvez avoir plus d'un capteur sur le même bus. 0 se réfère au premier capteur
//(DS18B20) sur le bus
 Serial.println(" °C");
 Serial.print("Temperature DHT11: ");
 Serial.print(DHT.temperature); //Afficher la température et humidité mesurée par DHT11
 Serial.println(" °C");
 Serial.print("Temperature NTC: ");
 Serial.print(Tc); //Afficher la température mesurée par la thermistance
 Serial.println(" °C");
 Serial.println(" ");
 delay(1000);
}
```

- **Explication et analyse du sketch :**

Le code du sketch contient 7 parties :

1- La première partie concerne l'incorporation des trois bibliothèques : 'OneWire', 'DallasTemperature' et 'dht', la déclaration et l'initialisation des variables ainsi que l'instanciation des objets.

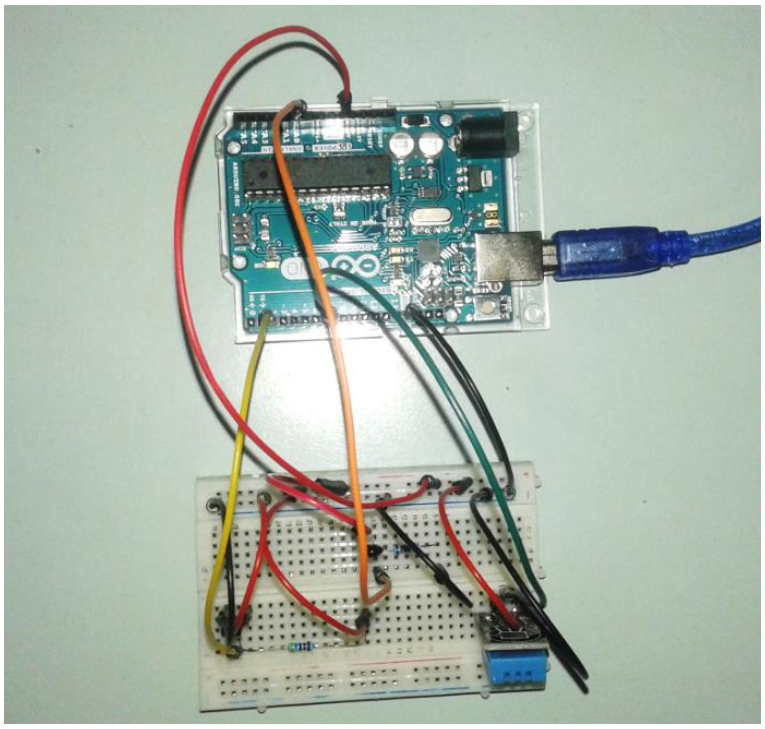
2- Dans le bloc de la fonction setup, nous avons défini la vitesse de transmission du moniteur série et initialisé le capteur DS18B20.

3- Dans cette partie, nous avons défini le code du programme permettant fe mesurer la température avec la thermistance en se basant sur la relation de Steinhart.

4- Dans cette 4ème partie Nous avons demandé une lecture de la température et humidité mesurées par le DHT11 à l'aide de la méthode 'read11'.

5- Nous demandé à DS18B20une de démarrer une mesure de température à l'aide de la méthode 'requestTemperatures'.

6- Dans cette dernière partie, les résultats de mesure de température relevée par les 3 capteurs sont affichés sur le moniteur série.



Réalisation du montage

```
COM7 (Arduino/Genuino Uno)

Temperature DHT11: 20.00 °C
Temperature NTC: 16.31 °C

temperature DS18b20: 20.37 °C
Temperature DHT11: 20.00 °C
Temperature NTC: 16.48 °C

temperature DS18b20: 20.37 °C
Temperature DHT11: 20.00 °C
Temperature NTC: 16.31 °C

temperature DS18b20: 20.37 °C
Temperature DHT11: 21.00 °C
Temperature NTC: 16.40 °C
```

Résultats de mesure des trois capteurs

D'après la figure ci-dessous, nous pouvons constater que la thermistance NTC manque de précision en comparaison avec les deux autres capteurs.

Application : Commande d'un ventilateur selon une consigne de température

L'objectif de cette application est la commande d'un moteur de ventilation 12v selon une consigne de température fixée par l'utilisateur, les spécifications fonctionnelles sont :

- La température T en °C est affichée sur module LCD 16*02.
- Lorsque T est inférieure à la consigne de limite, une LED verte est allumée et un message (''Ventilateur : OFF '') s'affiche sur le LCD.
- Si la limite de consigne est dépassée, la LED verte s'éteint, une LED rouge s'allume et le moteur ventilateur fonctionne et un message (''Ventilateur : ON '') s'affiche sur le LCD.
- Lorsque la température baisse jusqu'à ce qu'elle soit inférieure à 35°C, le moto ventilateur s'arrête. La LED rouge s'éteint et la LED verte s'allume à nouveau.

Matériels et composants nécessaires :

Aduino Uno

Breadboard

Capteur DS18B20

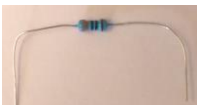
Fils de connexion

2* LED

-

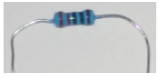
Afficheur LCD 16*02 avec interface standard 16 broches

Résistance 4.7 kΩ



2 * Résistance 330 Ω

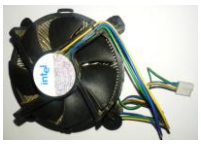
Potentiomètre 10 KΩ



Résistance 220 Ω

Résistance 1 kΩ

Transistor de puissance TIP120



Ventilateur 12v/0.4A

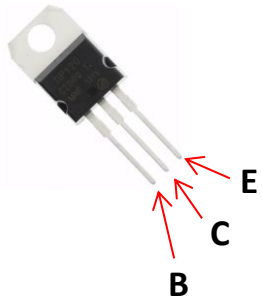
Adaptateur 12V DC/0.5A

Diode 1N4004

- **Réalisation du montage de l'application :**

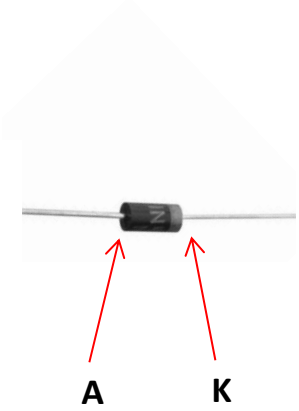
Circuit de commande du moto-ventilateur :

Le moteur du ventilateur utilisé a besoin de plus de courant et de tension que notre carte Arduino ne peut en fournir. Pour ce faire, nous allons utiliser un transistor de puissance de type TIP120. C'est un transistor Darlington en boitier To-220, capable de commuter un courant de collecteur Ic=5A et de supporter une tension collecteur-émetteur U_{CE} =60V. Aussi une diode de 'roue libre' est indispensable pour protéger le transistor (et la broche d'Arduino en liaison avec la base du transistor) d'une éventuelle surtension.



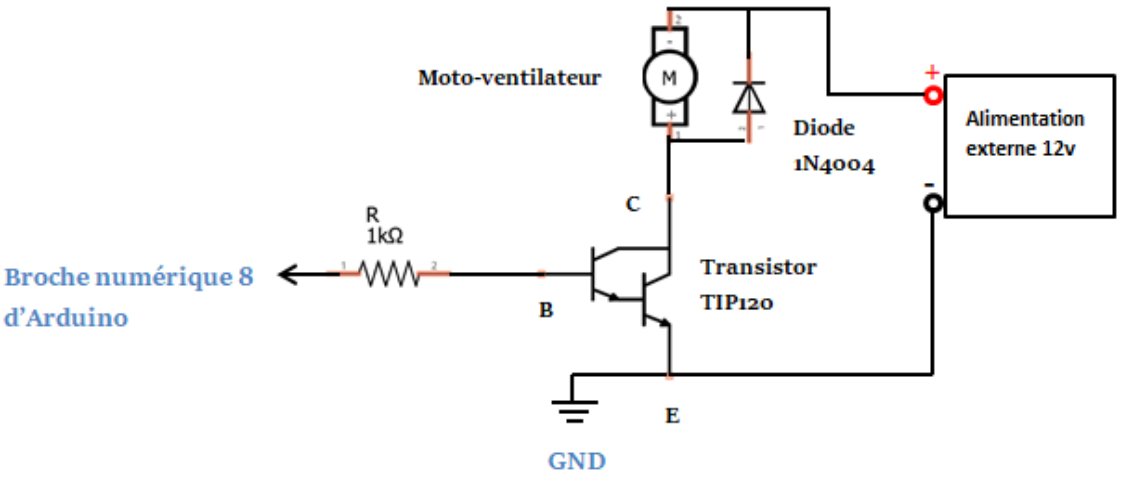
Transistor TIP120

Moto-ventilateur utilisé



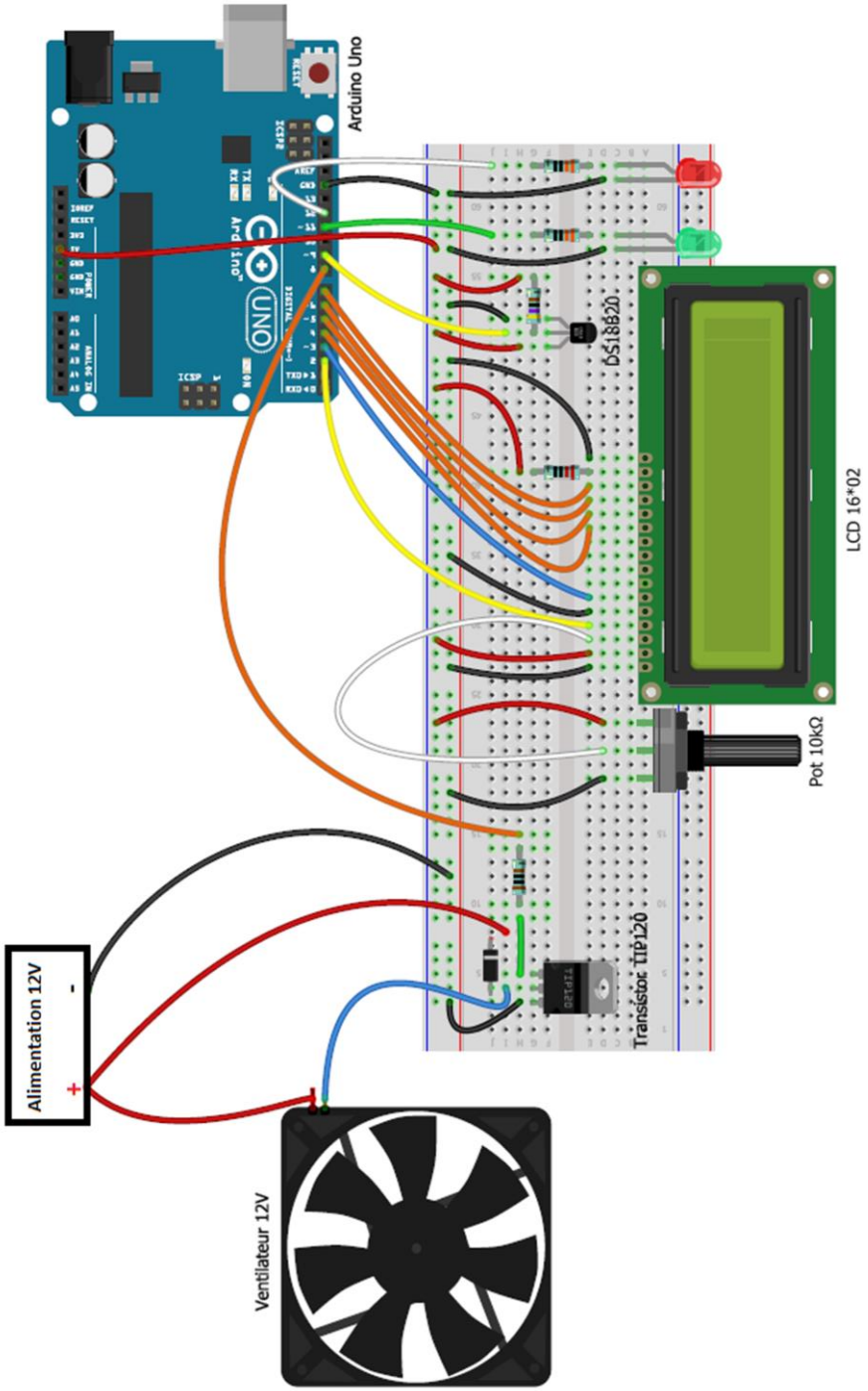
Diode 1N4004

Le circuit de commande du moto-ventilateur est illustré dans la figure ci-dessous :



Commande du moto-ventilateur avec un transistor de puissance TIP120

Montage de l'application :



Montage de l'application

- **Code du sketch :**

```
/*1-Incorpoation des bibliothèques, définition de constantes, déclaration des variables et instanciation des objets*/
#include <OneWire.h>
#include <DallasTemperature.h>
#include<LiquidCrystal.h>
#define ONE_WIRE_BUS  2 // Broche de données du bus oneWire connecté sur la broche 2
#define RS 2   // Register Select
#define E   3    // Enable
#define D4  4    // Ligne de données 4
#define D5  5    // Ligne de données 5
#define D6  6    // Ligne de données 6
#define D7  7    // Ligne de données 7
#define COL  16 // Nombre de colonnes
#define LIGNES  2  // Nombre de lignes
#define Trans  8
#define LedR  12
#define LedV  11
float T;
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);     // Instanciation de l'objet 'lcd'
OneWire oneWire(ONE_WIRE_BUS);  // Instanciation de l'objet 'oneWire'
DallasTemperature sensors(&oneWire); // Instanciation de l'objet ' sensors');//Transmettre //notre référence oneWire au capteur de température Dallas

void setup()
{
 /*2-Vitesse de transmission du moniteur série et, initialisation du module LCD et programmation des broches 8,11 et 12 en sorties*/
 lcd.begin(COL, LIGNES);    // Nombres de colonnes et de lignes
 sensors.begin(); //Initialiser le capteur DS18B20
 pinMode(LedV, OUTPUT); //Broche numérique 11 comme sortie
```

```
 pinMode(LedR, OUTPUT); //Broche numérique 12 comme sortie
 pinMode(Trans, OUTPUT); //Broche numérique 8 comme sortie
}

void loop(){
 getTemp();  //Apelle de la fonction 'getTemp' mesurer la température
 AfficherTlcd(); //Apelle de la fonction 'AfficherTlcd' pour afficher la température sur le LCD
 /*3-Traitement du cas où la température est inférieure ou égale à 35°C  */
 if(T<=35){
 lcd.setCursor(0,1);
 lcd.print("Ventilateur: OFF");
 digitalWrite(LedV, HIGH); //Allumer la LED verte
 digitalWrite(Trans, LOW);// Ventilateur OFF
 digitalWrite(LedR, LOW);//Eteindre la LED rouge
 delay(500);
}
/*4-Traitement du cas où la température est supérieure à 35°C  */
 else{
  lcd.clear();
  digitalWrite(LedV, LOW); // Eteindre la LED verte
  digitalWrite(Trans, HIGH); // Ventilateur ON
  digitalWrite(LedR, HIGH); // Allumer la LED rouge
  AfficherTlcd();//Afficher la température sur le module LCD
  lcd.setCursor(0,1);
  lcd.print("Ventilateur: ON");
  delay(500);
  getTemp(); //Actualiser la valeur de mesure de température
 }
}
/*5-Définition de la fonction 'getTemp' qui mesure la température  */
void getTemp(){
 sensors.requestTemperatures();  // Demande de mesure de température
```

```
 T=sensors.getTempCByIndex(0); //Stocker la valeur de mesure dans la variable 'T', vous
//pouvez avoir plus d'un capteur sur le même bus. 0 se réfère au premier capteur (DS18B20)
//sur le bus
 }
/*5-Définition de la fonction 'AfficherTlcd' qui affiche la température sur le module LCD  */
 void AfficherTlcd(){
 lcd.setCursor(0,0);
 lcd.print(" Temp: ");
 lcd.print(T);
 lcd.print(" ");
 lcd.print((char)223);//'223' :code ASCCI du caractère '°' (degré)
 lcd.print("C");
 }
```

- **Explication et analyse du sketch :**

Le code du sketch contient 7 parties :

1- La première partie concerne l'incorporation des trois bibliothèques : 'OneWire', 'DallasTemperature' et 'dht', la déclaration et l'initialisation des variables ainsi que l'instanciation des objets.

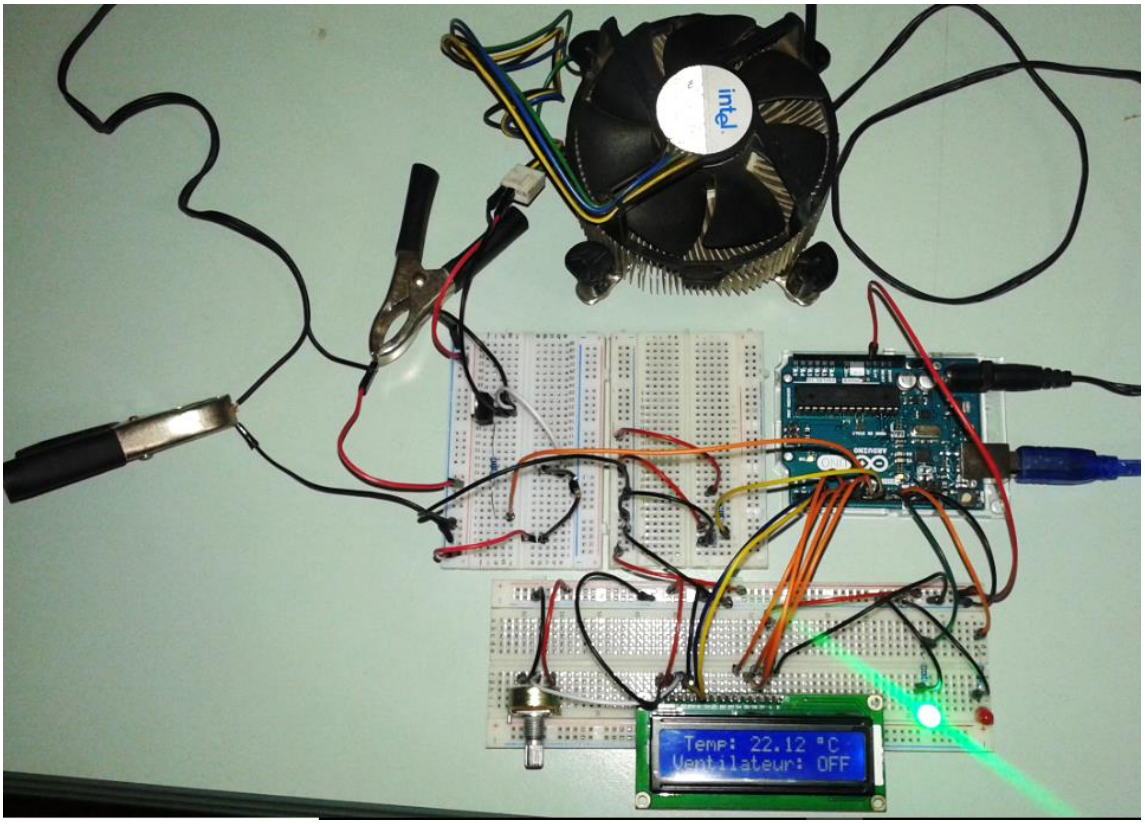
2- Au sein du bloc de la fonction setup, nous avons défini la vitesse de transmission du moniteur série et initialisé le capteur DS18B20 et l'afficheur LCD, ainsi que nous avons programmé les broches numériques 8,11 et 12 en sorties.

3- Dans cette partie, nous avons défini le code du programme permettant de traiter le cas où la température est inférieure ou égale à 35°C, dans lequel on allume la LED verte, on éteint la LED rouge et on s'assure que le ventilateur est en état d'arrêt.

4- Le cas où la température est supérieure à 35°C est traité dans cette partie, La LED verte doit être éteinte et la LED rouge allumée, ainsi que le ventilateur doit fonctionner.

5- Dans cette partie nous avons défini la fonction qui se charge de mesurer la température avec le capteur DS18B20.

6- Dans la dernière partie, partie nous avons défini la fonction qui affiche la température ainsi mesurée sur l'afficheur LCD.



Réalisation du montage

LEÇON 9

Le détecteur ultrasonique

Au sommaire de cette 8ème leçon :

Le détecteur ultrasonique :

Les capteurs à ultrasons sont de petits modules électroniques qui mesurent la distance. Vous pouvez les utiliser pour trouver la distance à un objet, ou pour détecter quand quelque chose se trouve près du capteur (détecteur de mouvement). Ils sont parfaits pour les projets impliquant la navigation, l'évitement d'objets et la sécurité à la maison. Ils fonctionnent aussi bien dans le noir que dans la lumière parce qu'ils utilisent les ondes ultrason pour mesurer la distance.

Le détecteur ultrasonique que nous allons utiliser dans cette leçon est le **HC-SR04**, qui peut mesurer des distances de **2 cm** à **400 cm** avec une précision de **± 3 mm**.

Détecteur ultrasonique HC-SR04

- **Principe du fonctionnement :**

Vitesse du son :

Les détecteurs à ultrasons mesurent la distance en émettant une impulsion de son ultrasonique qui se propage dans l'air jusqu'à atteindre un objet. Lorsque cette impulsion de son frappe un objet, elle est réfléchie par l'objet et retourne au détecteur ultrasonique. Ce dernier mesure le temps nécessaire pour que l'impulsion sonore se déplace dans son trajet aller-retour depuis le capteur et vers l'arrière. Il envoie ensuite un signal au microcontrôleur avec des informations sur le temps qu'il a fallu pour que l'impulsion sonore se déplace.

Connaissant le temps qu'il faut à l'impulsion ultrasonique pour aller frapper un objet et retourner au détecteur ultrasonique, et connaissant aussi la vitesse du son, le µc peut calculer la distance à l'objet. La formule qui relie la vitesse du son, la distance et le temps parcouru est :

$$\textbf{Vitesse} = \frac{\textbf{Distance}}{\textbf{Temps}}$$

En réorganisant la formule précédente, nous obtenons la formule utilisée pour calculer la distance :

$$\textbf{Distance} = \textbf{Vitesse} * \textbf{Temps}$$

La variable du temps est le temps nécessaire pour que l'impulsion ultrasonique sorte du capteur, rebondisse sur l'objet et retourne au capteur. Nous divisons en fait ce temps en deux puisque nous avons seulement besoin de mesurer la distance à l'objet, pas la distance à l'objet et le retour au capteur. La variable de vitesse est la vitesse à laquelle le son traverse l'air.

La vitesse du son dans l'air change avec la température et l'humidité. Par conséquent, afin de calculer la distance avec précision, nous devrons prendre en compte la température ambiante et l'humidité. La formule de la vitesse du son dans l'air en fonction de la température et l'humidité est :

$$\mathbf{C} = 331.4 + (0.606 * \mathrm{T}) + (0.0124 * \mathrm{H})$$

Avec :

C : Vitesse du son en mètres par seconde (m/s).

331.4 : Vitesse du son à 0°C et 0% d'humidité.

T : Température en °C.

H : % d'humidité (humidité relative).

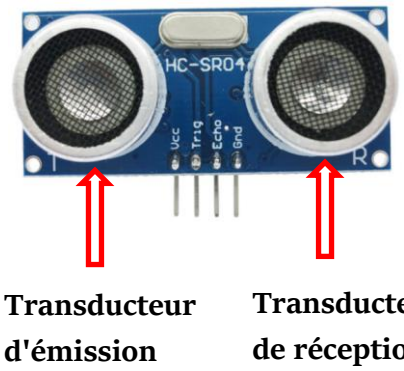
Par exemple, à 20 ° C et 50% d'humidité, le son se déplace à une vitesse de :

$$\mathbf{C = 331.4 + (0.606 * 20) + (0.0124 * 50) = 344.14\ m/s}$$

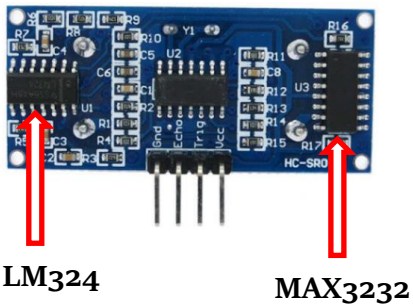
Dans l'équation ci-dessus, il est clair que la température a le plus grand effet sur la vitesse du son. L'humidité a une certaine influence, mais c'est beaucoup moins que l'effet de la température.

Comment le module HC-SR04 mesure-il la distance ?

Sur la face avant du module HC-SR04 (figure à gauche ci-dessous) sont fixés deux cylindres métalliques. Ce sont des transducteurs. Les transducteurs convertissent les forces mécaniques en signaux électriques. Dans un capteur à ultrasons, il y a un transducteur d'émission et un transducteur de réception. Le transducteur d'émission convertit un signal électrique en une impulsion ultrasonique, et le transducteur de réception convertit l'impulsion ultrasonique réfléchie en un signal électrique.



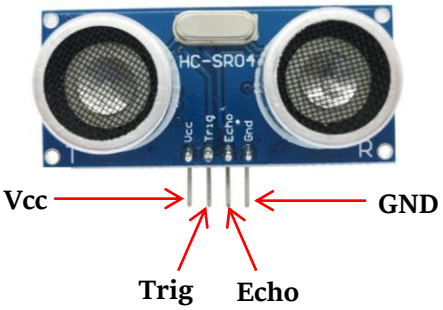
Face avant du module HC-SR04



Face arrière du module HC-SR04

Si vous regardez en arrière du module HC-SR04 (figure à droite ci-dessus), vous verrez un petit circuit intégré (CI) derrière le transducteur d'émission (**MAX3232**), c'est le CI qui contrôle le transducteur d'émission. Derrière le transducteur de réception se trouve un autre CI (**LM324**), il s'agit d'un quad ampli-op qui amplifie le signal généré par le transducteur de réception en un signal suffisamment puissant pour le transmettre au microcontrôleur.

Brochage du module HC-SR04 :



Les 4 broches du module HC-SR04

Le module HC-SR04 possède quatre broches (figure précédente) : **Vcc**, **Trig**, **Echo** et **GND**. La broche **Vcc** fournit l'énergie nécessaire pour générer les impulsions ultrasoniques. La broche **GND** doit être connectée à la masse. La broche **Trig** est l'endroit où le microcontrôleur envoie le signal pour démarrer l'impulsion ultrasonique et sur la broche **Echo**, le module HC-SR04 envoie au µc les informations sur la durée du trajet pris par l'impulsion ultrasonique.

Mesurer la distance à un objet avec le module HC-SR04 :

Pour lancer une mesure de distance avec le module HC-SR04, nous devons envoyer un signal de 5V (HIGH) à la broche **Trig** pendant au moins 10 µs. Lorsque le module reçoit ce signal, il émet 8 impulsions de son à ultrasons à une fréquence de 40 KHz par le transducteur d'émission. Ensuite, il attend et écoute sur le transducteur de réception si le signal va être réfléchi. Si un objet est dans la portée, les 8 impulsions seront renvoyées au capteur et lorsque l'impulsion frappe le transducteur de réception, la broche **Echo** émet un signal HIGH pendant une durée précise (selon la distance entre le module et l'objet), puis revient à l'état LOW.

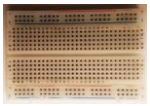
La durée du signal HIGH est égale au temps total que prennent les 8 impulsions pour se déplacer du transducteur d'émission et revenir au transducteur de réception. Cependant, nous voulons seulement mesurer la distance à l'objet, et non la distance du chemin que l'impulsion sonore a pris. Par conséquent, nous divisons ce temps en deux pour obtenir la variable temps dans l'équation **d = v x t** citée précédemment. Puisque nous connaissons déjà la vitesse du son (v), nous pouvons résoudre l'équation pour calculer la distance.

- **Afficher la distance à un objet sur le moniteur série avec le module HC-SR04:**

Commençons par faire un simple télémètre à ultrasons qui produira des mesures de distance à un objet sur votre moniteur série de l'IDE Arduino.

Matériels et composants nécessaires :

Aduino Uno



Breadboard

Fils de connexion

Module HC-SR04

Schéma du montage :

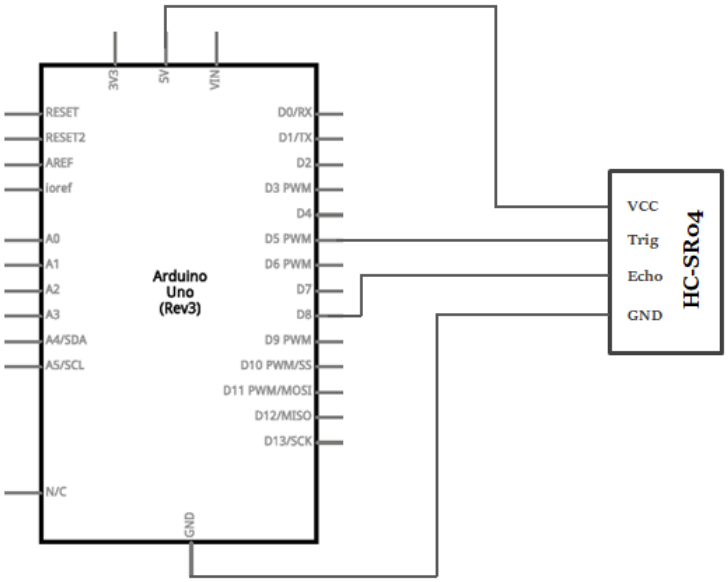
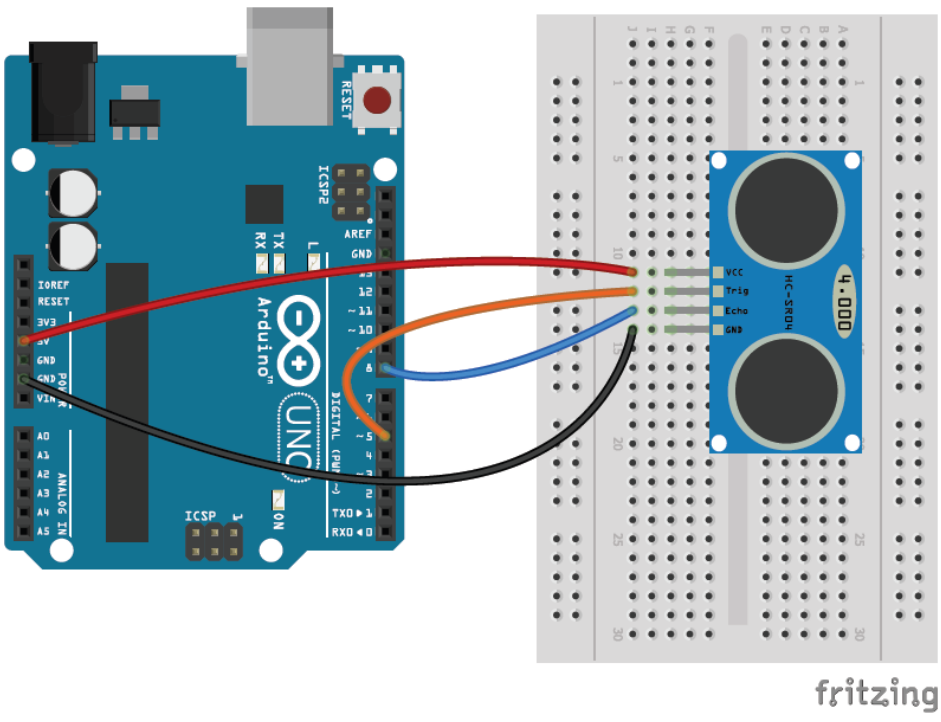


Schéma du montage



Réalisation du montage avec fritzing

Code du sketch :

```
#define trigBroche 5   //Broche Trig
#define echoBroche 8   //Broche Echo
float duree; //Durée du signal HIGH sur la broche Echo
float distance; //Distance entre le HC-SR04 et un objet

void setup() {
 Serial.begin (9600);  //Définition de la vitesse de transmission (baud)
 pinMode(trigBroche, OUTPUT); //Programmation de la broche 5 comme sortie
 pinMode(echoBroche, INPUT);  //Programmation de la broche 8 comme entrée
}

void loop() {
 digitalWrite(trigBroche, LOW);      //Mettre la broche 5 à l'état LOW
 delayMicroseconds(2);               //Attendre 2 microsecondes
 digitalWrite(trigBroche, HIGH);     //Mettre la broche 5 à l'état HIGH
 delayMicroseconds(10);              //Attendre 10microsecondes
 digitalWrite(trigBroche, LOW);      //Mettre la broche 5 à l'état HIGH
 duree = pulseIn(echoBroche, HIGH); //Affectation de la valeur renvoyée par la fonction
//'pulsIn' à la variable 'duree'
 distance = (duree / 2) * 0.0344;    //Calcul de la distance en cm/µs
 if (distance >= 400 || distance <= 2){  //Si la distance calculée est hors intervalle (2-400)cm
  Serial.print("Distance = ");
  Serial.println("Hors intervalle (2-400)cm");
 }
 else {
  Serial.print("Distance = ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(500);
 }
```

```
  delay(500);
}
```

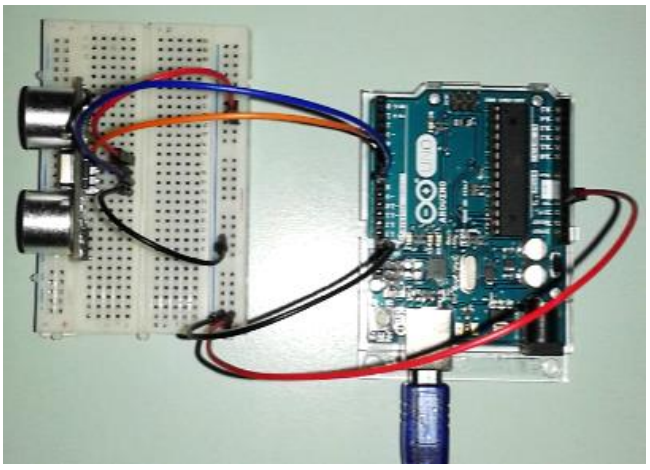
Explication et analyse du sketch :

En première partie, nous avons défini les deux broches numériques 5 et 8 pour les deux broches 'Trig' et 'Echo' du module HC-SR04, puis on a déclaré les deux variables 'duree' (longueur en µs de tout signal d'entrée HIGH détecté à l'entrée 'echoBroche') et 'distance' (distance entre le module HC-SR04 et l'objet, c'est le temps (t) multiplié par la vitesse du son convertie du mètres par seconde en centimètres par µs (0,0344 cm / µs).

Dans le bloc de la fonction setup, nous avons défini la vitesse de transmission du moniteur série, ainsi que nous avons programmé la broche 5 (Trig) comme entrée et la broche 8 (Echo) comme sortie.

Au début de la fonction loop, nous envoyé un signal LOW de 2 µs à la broche 'Trig' pour s'assurer qu'elle est désactivée au début de la boucle du programme. Puis on lui a envoyé un signal HIGH de 10 µs pour déclencher la séquence de huit impulsions ultrasoniques de 40 kHz émises par le transducteur d'émission.

La fonction pulseIn attend que la broche 'Echobroche' passe au niveau HIGH, commence le chronométrage, puis attend que cette broche passe à LOW et arrête le chronométrage, et elle renvoie la longueur de l'impulsion en microsecondes (duree). Si la mesure de distance n'est pas hors intervalle (2-400)cm, on affiche alors la distance calculée sur le moniteur série pendant 500 ms. Sinon on affiche le message "Hors intervalle (2-400)cm".



Réalisation du montage

Application : Réalisation d'un télémètre sonore

Dans cette application, nous allons réaliser un télémètre performant à base du module HC-SR04, le montage de l'application va nous permettre d'obtenir le fonctionnement suivant :

- Un module LCD 16*02 affiche la distance entre le HC-SR04 et un objet qui lui approche.
- Si la distance est supérieure à 20 cm (distance limite), une LED verte s'allume.
- Si la distance est inférieure ou égale à 20 cm, la LED verte s'éteint et une deuxième LED rouge se met à clignoter avec une vitesse dépendante de la distance entre l'objet et le capteur ultrasonique (si la distance décroit, la vitesse de clignotement accélère, et si la distance augmente, la vitesse de clignotement ralentit). De la même façon et en même temps, un buzzer actif déclenche une série de pipes avec un message(" Objet proche") qui s'affiche sur le module LCD.

Puisque la température et l'humidité sont deux variables de la vitesse de l'équation sonore (**c = 331,4 + (0,606 x T) + (0,0124 x H)**), la température et l'humidité de l'air autour du capteur HC-SR04 affecte nos mesures de distance. Pour compenser cela, Nous allons exploiter la mesure des deux paramètres (température et humidité) avec le capteur DHT11.

Matériels et composants nécessaires :

Aduino Uno

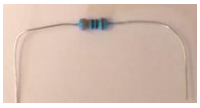
Breadboard

Module HC-SR04

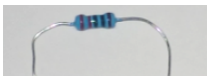
Fils de connexion

Afficheur LCD 16*02 avec interface standard 16 broches

Module DHT11



2 * Résistance 330 Ω



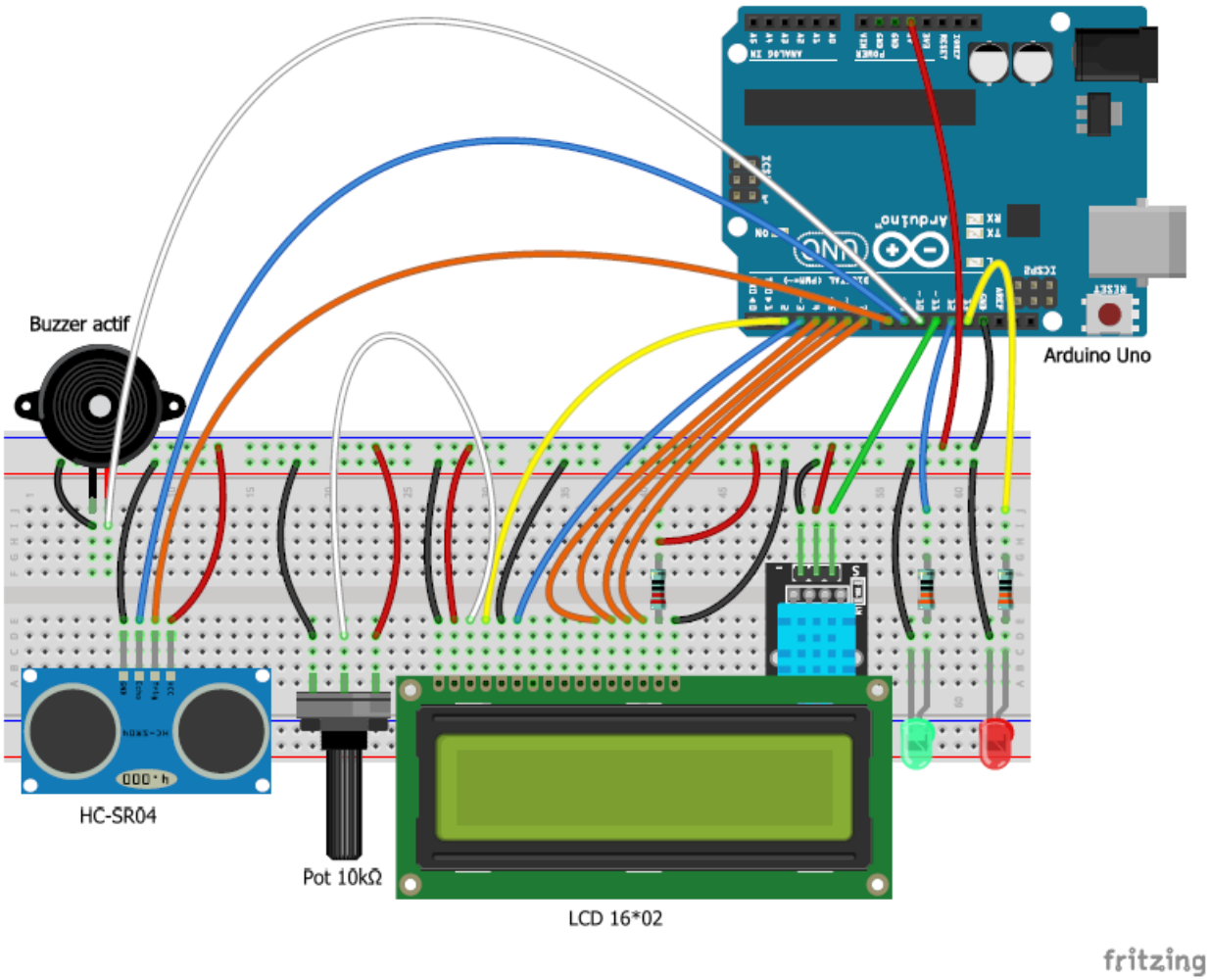
Résistance 220 Ω

Potentiomètre 10 KΩ

Buzzer actif

2* LED

- **Montage de l'application :**



Réalisation du montage avec fritzing

- **Code du sketch :**

Dans ce sketch nous allons utiliser la bibliothèque 'dht'. Pour installer cette bibliothèque, Rendez-vous sur la barre de menu de l'IDE Arduino, puis accédez à **Croquis** > **Inclure une bibliothèque** > **Gérer les bibliothèques** et cherchez "dht". Cliquez sur la bibliothèque, puis cliquez sur **installer**.

```
/*1- Définition des constantes, déclaration et initialisation des variables*/
#include <dht.h>
#include <LiquidCrystal.h>
#define COL 16
#define LIGNES 2
```

```
#define RS 2  // Register Select
#define E 3   // Enable
#define D4  4  // Ligne de données 4
#define D5  5  // Ligne de données 5
#define D6  6  // Ligne de données 6
#define D7  7  // Ligne de données 7
#define trigBroche  8   //Broche Trig du module HC-SR04
#define echoBroche  9   //Broche Echo du module HC-SR04
#define buzzer 10  //Broche du buzzer actif
#define DHT11_Broche  11  //Broche du capteur DHT11
#define LedVerte  12  //Broche de la LED verte
#define LedRouge  13  //Broche de la LED rouge
float duree;
float distance;
float vitesse;
int tempo;
int dhtRes;    //Variable pour stocker la valeur de lecture de la broche du capteur DHT11
/*2- Instanciation des objets lcd et dht*/
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);  // Instanciation de l'objet lcd
dht DHT;  // Instanciation de l'objet dht
/*3- Programmation de broches numériques*/
void setup() {
 lcd.begin(COL, LIGNES);   // Nombres de colonnes et de lignes
 pinMode(trigBroche, OUTPUT); //Programmation de la broche 8 comme sortie
 pinMode(echoBroche, INPUT);  //Programmation de la broche 9 comme entrée numérique
 pinMode(buzzer, OUTPUT);    //Programmation de la broche 10 comme sortie
 pinMode(LedVerte, OUTPUT);   //Programmation de la broche 12 comme sortie
 pinMode(LedRouge, OUTPUT);   //Programmation de la broche 13 comme sortie
}

void loop() {
/*4- Programme traitant le cas d'une distance supérieure à 20cm*/
```

```
  getdistance(); //Calculer la distance entre l'objet et le HC-SR04 en appelant 'getdistance'
 if (distance > 20 && distance<400){
   digitalWrite(LedVerte, HIGH); //Allumer la LED verte
   lcd.print("   Distance :   ");
   lcd.setCursor(4,1);           //Positionner le curseur sur la 4ème colonne de la 1ère ligne
   lcd.print(distance);
   lcd.print(" cm");
   delay(500);  //Attendre 500 ms
 }
/*5- Programme traitant le cas d'une distance inférieure ou égale à 20cm*/
 else {
  while(distance <= 20 ){ //Tant que la distance est inférieure ou égale à 20
   digitalWrite(LedVerte, LOW); //Eteindre la LED verte
   lcd.print("  Objet proche  ");
   lcd.setCursor(4,1);   //Positionner le curseur sur la 4ème colonne de la 2ère ligne
   lcd.print(distance);
   lcd.print(" cm");
   tempo=map(distance,3,20,0,700);//Affectation de la valeur renvoyée par la fonction map à
//la variable tempo
   digitalWrite(buzzer, HIGH); //Déclencher  l'alarme du buzzer
   digitalWrite(LedRouge, HIGH);//Allumer la LED rouge
   delay(tempo); //Attendre (tempo ms)
   digitalWrite(buzzer, LOW);//Désactiver l'alarme du buzzer
   digitalWrite(LedRouge, LOW); //Eteindre la LED rouge
   delay(tempo);
   getdistance();
   lcd.clear();  //Effacer l'afficheur LCD
   }
 }
 delay(500);
 lcd.clear();
```

```
}
/*6- Définition de la fonction de clacul de la distance 'getdistance'*/
void getdistance(){
 digitalWrite(trigBroche, LOW); //Mettre la broche 'Trig' sur le niveau HIGH
 delayMicroseconds(2); //Attendre 2 µs
 digitalWrite(trigBroche, HIGH); //Mettre la broche 'Trig' sur le niveau HIGH
 delayMicroseconds(10); //Attendre 10 µs
 digitalWrite(trigBroche, LOW); //Mettre la broche 'Trig' sur le niveau LOW
 duree = pulseIn(echoBroche, HIGH);//Affectation de la valeur renvoyée par la fonction
//'pulsIn' à la variable 'duree'
 dhtRes = DHT.read11(DHT11_Broche); //Affectation de la valeur renvoyée par la méthode
//read11 à la variable 'dhtRes'
 vitesse = 331.4 + (0.606 * DHT.temperature) + (0.0124 * DHT.humidity); //Calcul de la
//vitesse du son
 distance = (duree / 2) * (vitesse / 10000); //Calcul de la vitesse en cm/µs
}
```

- **Explication et analyse du sketch :**

Le code du sketch peut être divisé en 9 parties :

1- Cette première partie, concerne l'incorporation des deux bibliothèques «LiquidCrystal » et « dht », la définition des différentes constantes et la déclaration et initialisation des variables qui seront utilisé dans le sketch.

2- Dans la 2ème partie, nous avons instancié les deux objets : l'objet lcd de la classe **LiqudCrystal** et l'objet DHT de la classe **dht.**

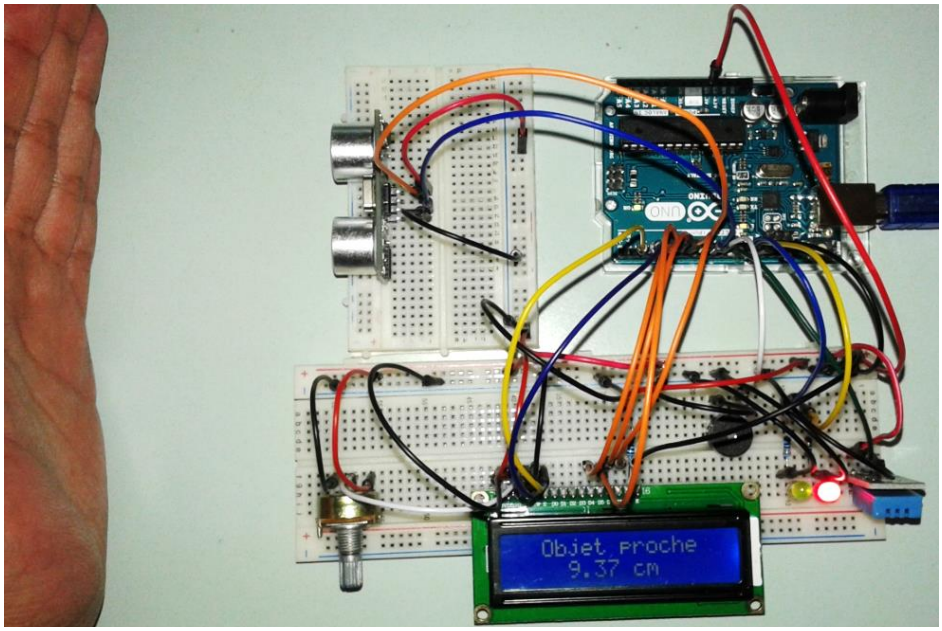
3- La 3ème partie consiste à programmer les différentes broches utilisées en sorties numériques (sauf la broche 9 (Echo) qui est utilisée comme entrée), ainsi que la définition du nombre de lignes et de colonnes de l'afficheur LCD.

4- Cette partie du sketch traite le cas où la distance entre un objet et le capteur ultrasonique est supérieure à 20 cm. On appelle d'abord la fonction getdistance() pour

calculer la distance, si le résultat renvoyé par cette fonction est compris entre 20 et 400, on allume alors la LED verte et on affiche cette valeur sur l'afficheur LCD.

5- Dans cette partie nous avons défini le code permettant de traiter le cas d'une distance inférieure ou égale à 6. On éteint premièrement la LED verte et on affiche cette distance sur le module LCD, puis, avec la fonction map on transpose le domaine de valeurs (3-20 cm) dans le domaine (0-700 ms). Et on affecte la nouvelle valeur renvoyée par map à la variable 'tempo'. Le contenu de la variable 'tempo' est utilisé comme retard pour faire clignoter la LED rouge et faire piper le buzzer actif. Et à la fin de cette partie on actualise le contenu de la variable 'distance' en calculant ça valeur par un nouveau appelle de la fonction getdistance() et on efface le LCD pour afficher la nouvelle valeur.

6- Nous avons défini dans cette dernière partie le code de la fonction getdistance() qui nous permet de calculer la distance. En se basant sur la formule de l'équation sonore : **c = 331,4 + (0,606 x T) + (0,0124 x H)**, les valeurs de mesure de la température (°C) et de l'humidité (%) sont contenues dans les deux attributs 'temperature' et 'humidity' de l'objet DHT. On calcule alors la vitesse du son **c** et on déduit la valeur de la distance en cm avec la formule : **distance = (duree / 2) * (vitesse / 10000)**.



Réalisation du montage

LEÇON 10

Le servomoteur

Au sommaire de cette 10ème leçon :

Le servomoteur :

Le servomoteur (souvent abrégé en « servo ») est un petit moteur à courant continu équipé de trois broches et dont le positionnement est commandé par des impulsions électriques (modulation de largeur d'impulsions : MLI), ces impulsions indiquent au servo la position à laquelle il doit se diriger.

Servomoteur 5V

Symbole d'un servomoteur

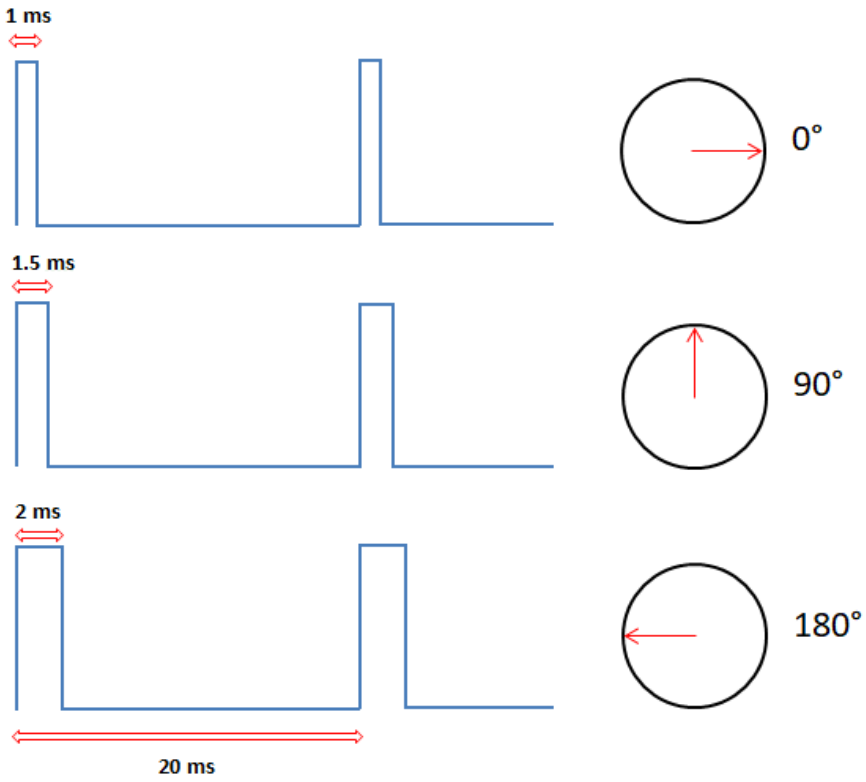
Le servomoteur intègre dans un même boitier, la mécanique (moteur et engrenage), et l'électronique, pour la commande et l'asservissement du moteur. Si le sevo est non modifié, la position est définie avec une limite de débattement d'angle de 180 degrés.

Les applications des servomoteurs sont diverses, ils sont généralement utilisés pour piloter les avions télécommandés en ajustant les volets d'aile, la position de vol des drones, les vannes de contrôle utilisées dans le contrôle du débit ou l'entraînement continu des roues pour les robots. Ils peuvent être utilisés pour positionner ou ajuster presque tout ce que vous pouvez penser.

- **Principe de la commande MLI d'un servomoteur :**

Les servomoteurs sont commandés par l'intermédiaire d'un câble électrique à trois fils qui permet d'alimenter le moteur et de lui transmettre des consignes de position sous forme d'un signal codé en largeur d'impulsion. Cela signifie que c'est la durée des impulsions qui détermine l'angle absolu de l'axe de sortie et donc la position du bras de commande du servomoteur. La durée de la période du signale de commande étant 20 ms et la durée de l'impulsion doit être comprise entre 1 'butée de droite) et 2 ms (butée de gauche). La figure

ci-dessous montre trois positions de servomoteur avec les signaux périodiues de commande correspondants :

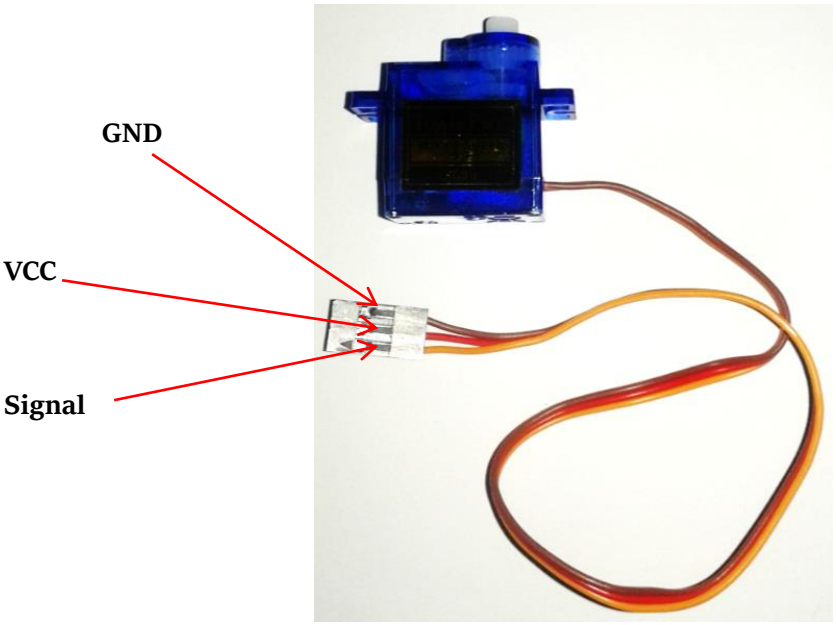


Commande MLI de la position d'un servomoteur avec une impulsion comprise entre 1 et 2 ms

1- Avec une durée d'impulsion de 1 ms, le servomoteur est amené sur la butée de droite qui correspond à un angle de 0°.
2- Avec une impulsion de 1.5 ms, le servomoteur peut prendre la position du milieu qui correspond à un angle de 90°.
3- Avec une durée d'impulsion de 2 ms, le servomoteur est amené sur la butée gauche (limite) qui correspond à un angle de 180°.

- **Brochage du servomoteur :**

Un servomoteur a trois fils, le fil marron est GND, le rouge est VCC, et l'orange est la ligne de signal.



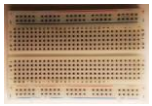
Brochage du servomoteur

- **Commander un servomoteur avec un potentiomètre :**

Dans cette partie, nous allons réaliser un montage permettant de varier l'angle de position de l'arbre du servomoteur de [0° à 180°] et de [180° à 0°] en agissant sur le curseur d'un potentiomètre.

Matériels et composants nécessaires :


Aduino Uno


Breadboard


Potentiomètre 10 KΩ


Servomoteur


Fils de connexion

- **Schéma du montage :**

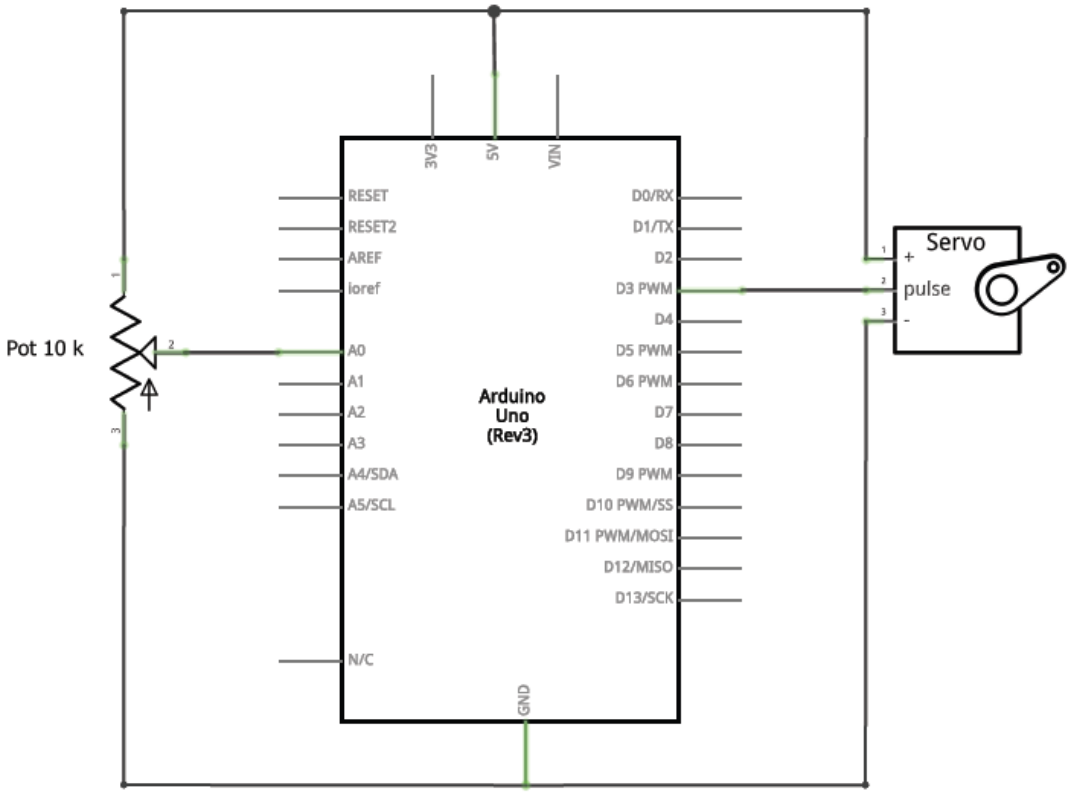
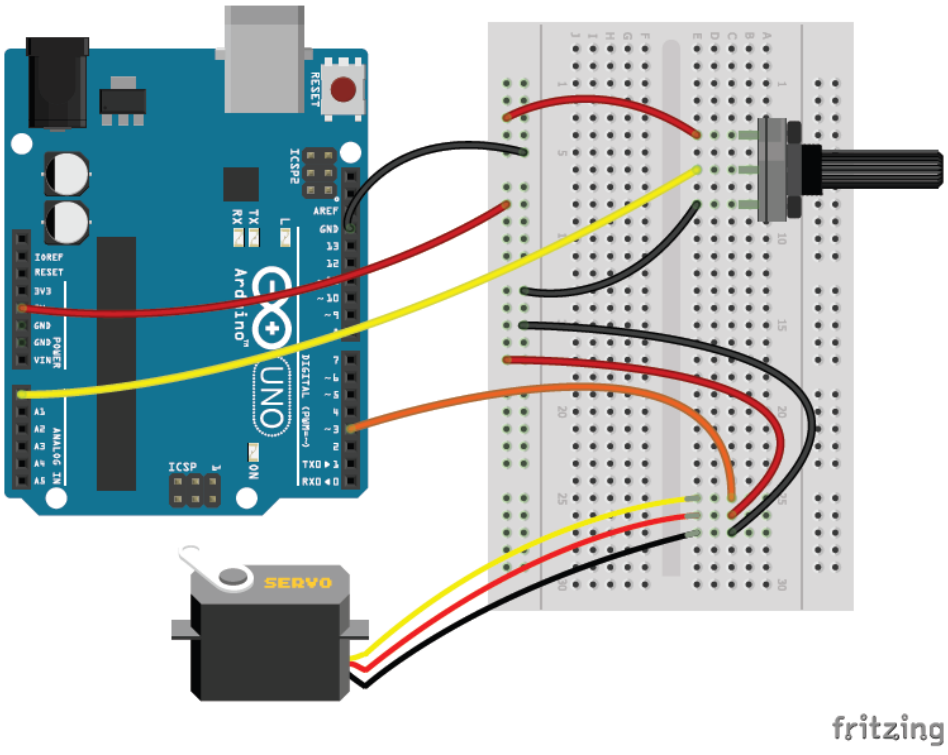


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

Dans ce sketch nous allons utiliser la bibliothèque 'Servo'. Pour installer cette bibliothèque, Rendez-vous sur la barre de menu de l'IDE Arduino, puis accédez à **Croquis** > **Inclure une bibliothèque** > **Gérer les bibliothèques** et cherchez "Servo". Cliquez sur la bibliothèque, puis cliquez sur **installer**.

```
#include <Servo.h>
int potBroche = 0;  // Broche analogique utilisée pour connecter le potentiomètre
int val;    // variable pour stocker la valeur de lecture de la broche analogique 0
Servo monservo;  // Instanciation d'un objet monservo

void setup() {
 monservo.attach(3);  // Attacher l'objet monservo à la broche numérique 3
}

void loop() {
 val = analogRead(potBroche);     // Affecter la valeur de lecture de l'entrée analogique 0
//valeur entre 0 et 1023)à la variable 'val'
 val = map(val, 0, 1023, 0, 180);     // Affectation de la valeur renvoyée par la fonction map à
'val'
 monservo.write(val);    // Régler l'angle de position du servo à 'val'°
 delay(15);                  // Attendre 15 ms
}
```

- **Explication et analyse du sketch :**

En premier temps, nous avons incorporé la bibliothèque «Servo » déclaré et initialisation les deux variables : 'potBroche' (Broche analogique utilisée pour connecter le potentiomètre) et 'val' (Pour stocker la valeur de lecture de la broche analogique o). Ensuite nous avant instancié un objet appelé 'monservo' de la classe Servo.

Dans la fonction setup, l'instruction `monservo.attach(3);` permet d'attacher l'objet 'monservo' à la broche numérique 3 (broche PWM).

Dans le bloc de la fonction loop, nous avons défini le code permettant de régler la position du servomoteur en variant le curseur du potentiomètre :

```
val = analogRead(potBroche);
```

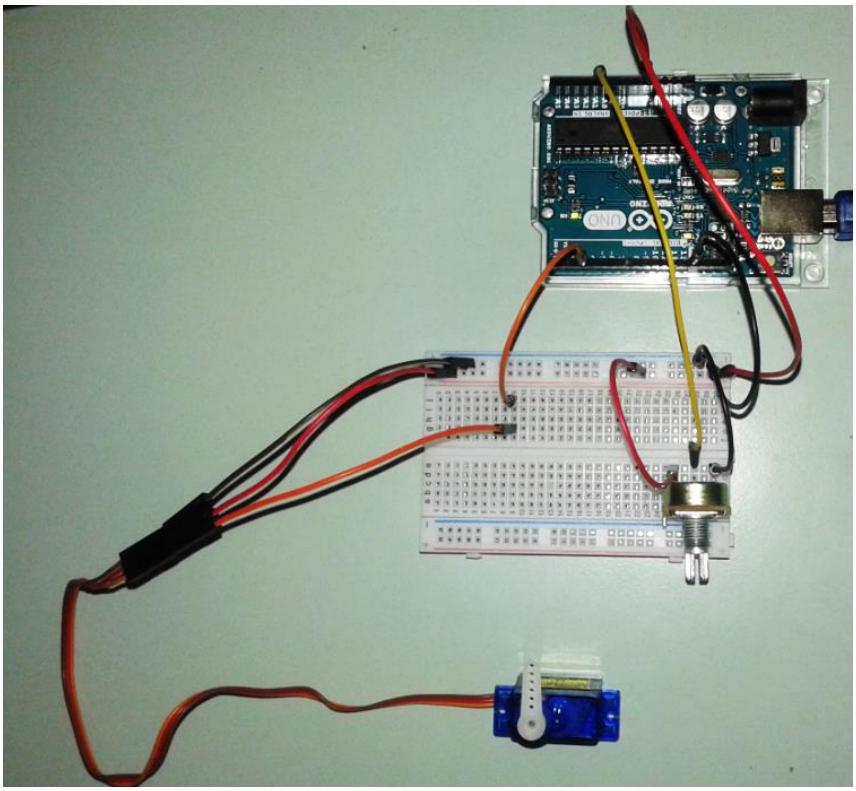
On lit d'abord la valeur du potentiomètre à l'entrée analogique 0 (valeur entre 0 et 1023), puis on stocke cette valeur dans la variable 'val'.

```
val = map(val, 0, 1023, 0, 180);
```

avec la fonction map on transpose le domaine de valeurs (0-1023) dans le domaine (0-180). Et on affecte la nouvelle valeur renvoyée par map à la variable 'val'. Le contenu de la variable 'val' est utilisé comme valeur de l'angle de position du servomoteur.

```
monservo.write(val);
```

La méthode write de la classe Servo qui accepte un seul paramètre (l'angle de position), permet de régler l'angle de position du servomoteur à une valeur bien précise.



Réalisation du montage

Application : Télémètre 140°

Le but de cette application est de réaliser un télémètre qui peut mesurer la distance à des objets présents partout sur une surface d'angle de 160°. Le capteur ultrasonique (HC-SR02) mesurant la distance sera entrainé par le servomoteur afin de pouvoir retourner de [20° à 180°] et de [180° à 20°] et pour signaler la présence d'un objet obstacle, un buzzer actif déclenchera une alarme si la distance entre le capteur et l'objet est inférieure à 6 cm.

Cette application peut être implantée, par exemple, sur un petit robot de véhicule afin de distinguer les obstacles sur le trajet de son mouvement.

Matériels et composants nécessaires :


Aduino Uno


Breadboard


Module HC-SR04


Servomoteur


Fils de connexion


Buzzer actif

- **Schéma du montage :**

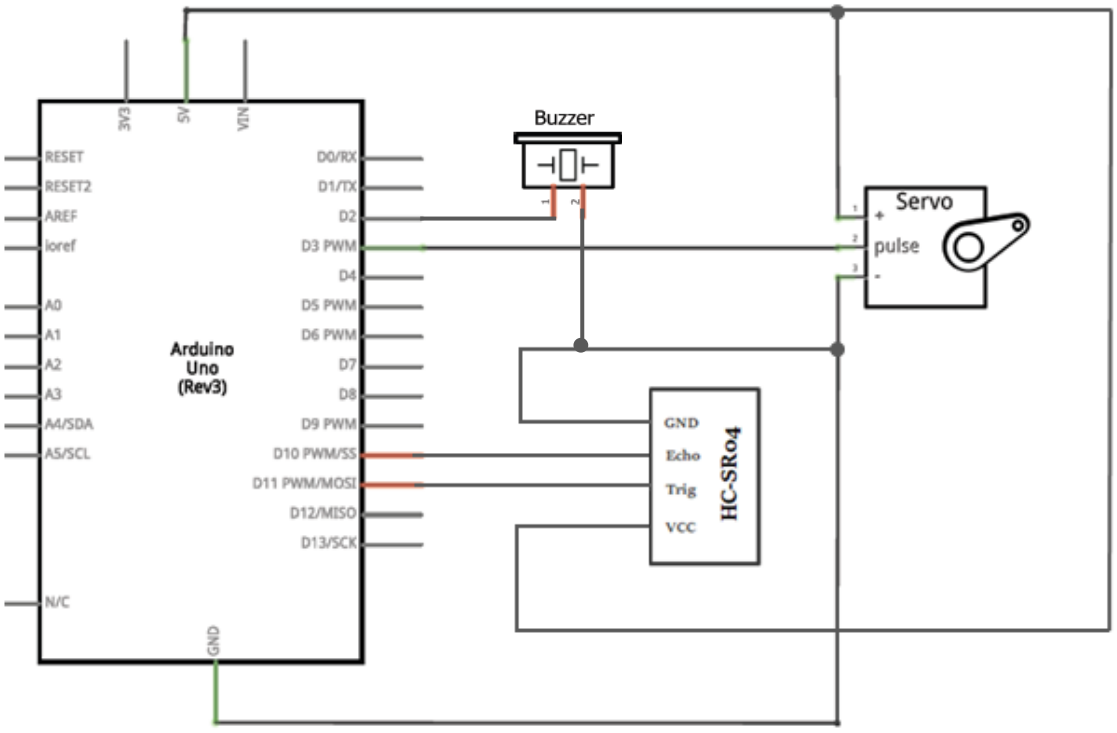
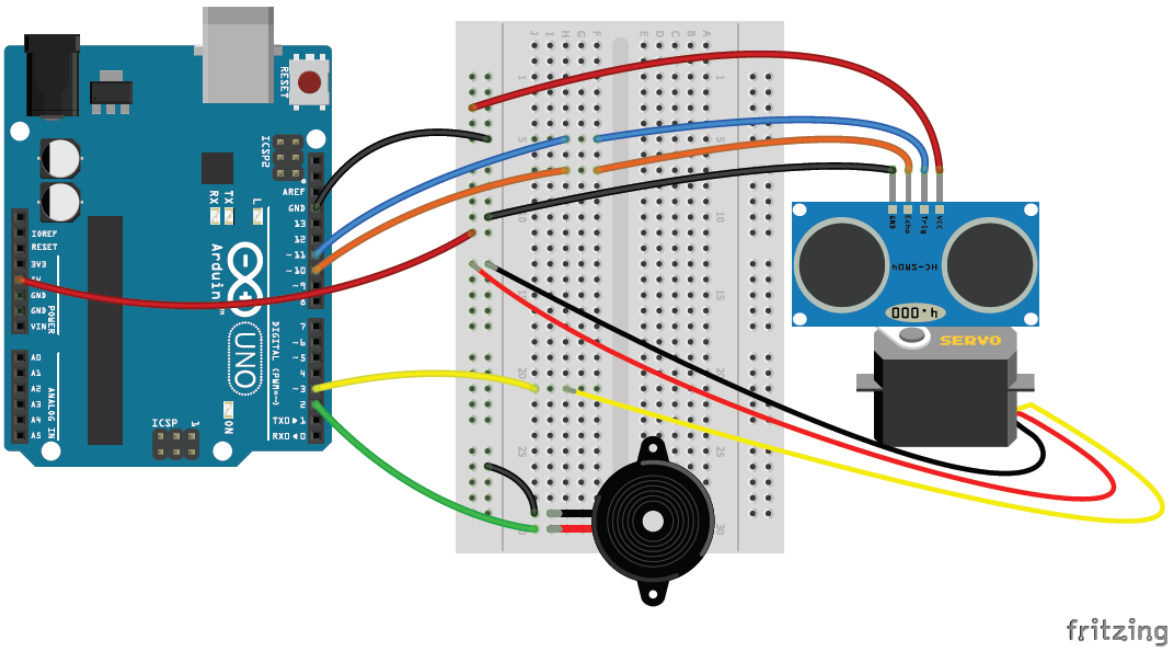


Schéma du montage



Réalisation du montage avec fritzing

- **Code du sketch :**

```
/* 1- Définition des constantes et déclaration des  variables*/
#include <Servo.h>
#define trigBroche 11   //Broche Trig
#define echoBroche 10   //Broche Echo
#define buzzer 2  //Broche du buzzer actif
float duree; //Durée du signal HIGH sur la broche Echo
float distance; //Distance entre le HC-SR04 et un objet
int pos = 0;    // variable to store the servo position
unsigned int tempo=300;
Servo  monservo;  // Instanciation de l'objet 'monservo'

void setup() {
/* 2- Programmation des broches numériques utilisées*/
monservo.attach(3);  // Attacher l'objet monservo à la broche numérique 3
 pinMode(trigBroche, OUTPUT); //Programmation de la broche 5 comme sortie
 pinMode(echoBroche, INPUT);  //Programmation de la broche 8 comme entrée
 pinMode(buzzer, OUTPUT);    //Programmation de la broche 10 comme sortie
```

```
}
void loop() {
/* 3- Mouvement du servomoteur de 20° à 180°*/
 for (pos = 20; pos <= 180; pos++) { // Aller de 20° à 180°
  monservo.write(pos);            // Régler l'angle de position du servo à 'pos'°
  delay(20);    //Attendre 20 ms
  getdistance();  //Appeler la fonction 'getdistance' pour calculer la distance entre le capteur
//ultrasonique et un objet proche
  testObstacle(); //Appeler la fonction 'testObstacle'
 }
/* 4- Movement du survomoteur de 180° à 20°*/
 for (pos = 180; pos >= 20; pos--) {   // Aller de 180°à 20°
  monservo.write(pos);            // Régler l'angle de position du servo à 'pos'°
  delay(20);    //Attendre 20 ms
  getdistance();
  testObstacle();
 }
}
/* 5- Définition de la fonction 'getdistance'*/
void getdistance(){     //Définition de la fonction getdistance()
 digitalWrite(trigBroche, LOW);   //Mettre la broche 'Trig' sur le niveau LOW
 delayMicroseconds(2);    //Attendre 2 µs
 digitalWrite(trigBroche, HIGH);    //Mettre la broche 'Trig' sur le niveau HIGH
 delayMicroseconds(10);    //Attendre 10 µs
 digitalWrite(trigBroche, LOW);    //Mettre la broche 'Trig' sur le niveau LOW
 duree = pulseIn(echoBroche, HIGH);   //Affectation de la valeur renvoyée par la fonction
//'pulsIn' à la variable 'duree'
 distance = (duree / 2) * 0.0344;    //Calcul de la distance en cm/µs
}
/* 6- Définition de la fonction 'testObstacle'*/

void testObstacle(){
```

```
  while(distance <= 6 ){    //Tant que la distance est inférieure ou égale à 20
   digitalWrite(buzzer, HIGH);    //Déclencher l'alarme du buzzer
   delay(tempo);    //Attendre (tempo ms)
   digitalWrite(buzzer, LOW);    //Désactiver l'alarme du buzzer
   delay(tempo);
   getdistance();    //Actualiser la valeur de la variable 'distance'
   }
}
```

- **Explication et analyse du sketch :**

Le code du sketch est organisé en 6 parties :

1- La première partie concerne la déclaration et l'initialisation des variables ainsi que instanciation d'un objet Servo.

2- Dans le bloc de la fonction setup, nous avons programmé les différentes broches numériques utilisées, aussi nous avons attaché l'objet créé 'monservo' à la broche numérique 3.

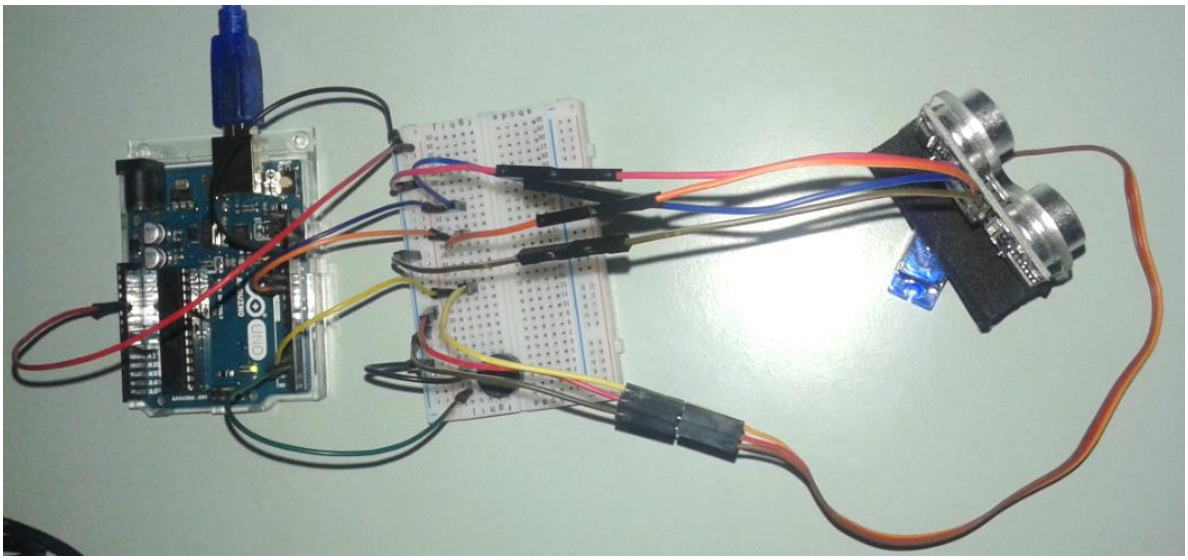
3- Dans cette partie, nous avons défini le code du programme permettant de faire bouger le servomoteur de 20° à 180°, de mesurer la distance à un objet proche et déclencher une alarme si cette distance est inférieure ou égale à 6. Le processus de traitement est géré par une boucle 'for'.

4- Dans cette 4ème partie et de la même façon que la dernière, nous avons utilisé une 2ème boucle 'for' afin de gérer le fonctionnement en cours du mouvement du servomoteur de 180° à 20°

5- Nous avons défini dans cette partie le code de la fonction getdistance() qui nous permet de calculer la distance entre l'objet et le capteur ultrasonique.

6- La 6ème et dernière partie, la définitition de la fonction 'testObstacle' qui nous permet de déclencher l'alarme du buzzer si la distance est <= 3cm.

N.B : pour plus d'informations sur le mode de fonctionnement du module HC-SR04, consultez-vous la leçon n°9.



Réalisation du montage