**Concevez des claviers pour l'Arduino**

Dans le chapitre précédent, vous avez avancé dans la programmation d’interfaces homme/machine en créant un afficheur avec quelques diodes. L'afficheur, quel que soit son type, est considéré comme une interface de sortie (l'Arduino produit de l'information).

Vous allez découvrir ici comment créer une interface pour des entrées, c'est-à-dire comment envoyer de l'information vers la carte Arduino, et ce sans avoir besoin de la connecter à un ordinateur.

En fait, nous allons tout simplement créer un clavier de saisie. Là encore, il ne s'agit pas d'un clavier complet (quoique vous pourriez le faire par la suite) mais d'un clavier de quelques boutons.

Nous allons pour ce faire étudier deux techniques différentes : la matrice de boutons, et la série de boutons.

Pour suivre ce chapitre, il vous faut prévoir :

* 4 boutons poussoir
* 2 diodes PH4148 ou 1N4148
* 2 résistances 10 KΩ
* 3 résistances 220Ω

**Concevez et utilisez une matrice de boutons**

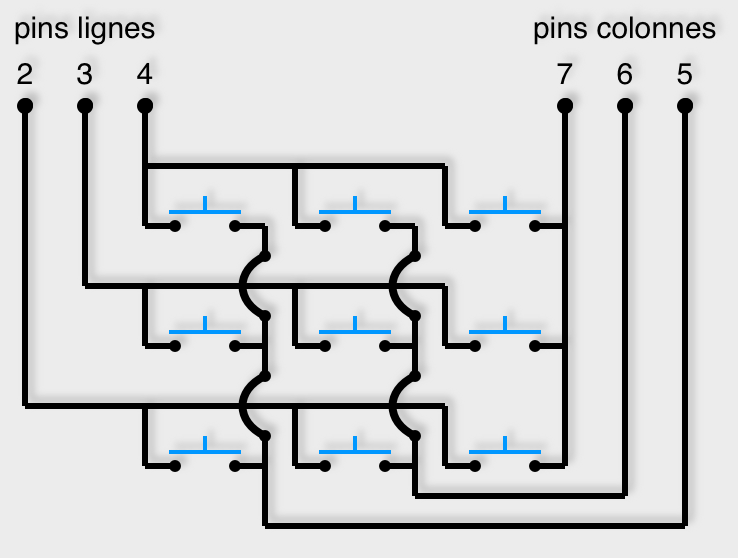
Qu'est-ce qu'une matrice de boutons ?

(Non Lukas, rien à voir avec l'acné, bien que dans votre cas, en effet on pense tout de suite à un tableau de bord d'avion)

Ce nom devrait vous évoquer quelque chose en rapport avec le chapitre précédent.

En effet, le principe va correspondre aux matrices de LED. Nous allons créer une sorte de grille de boutons que l'Arduino va parcourir comme précédemment, à l'aide d'un tableau.

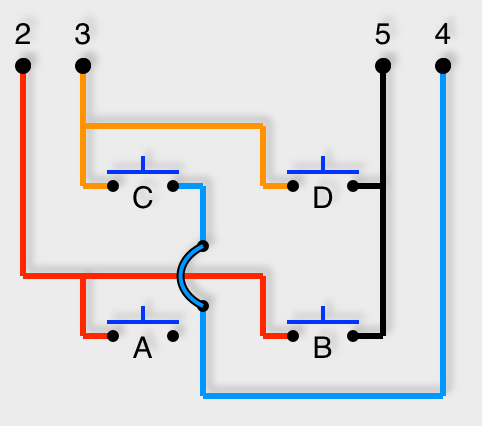
Voici donc le schéma qui correspond :

Matrice de 3x3 boutons poussoir

La matrice ci dessus comporte 9 boutons sur 3 lignes et trois colonnes. Je vous l'ai mis à titre indicatif pour pouvoir faire le parallèle avec le chapitre précédent.

Pour ce cours, et surtout pour des raisons à la fois techniques (montage) et financières (coût par bouton)  et sans gêner la théorie, nous allons seulement utiliser une matrice de boutons de 2 x 2 soit 4 boutons.

Voici donc le schéma théorique (les couleurs ont de l'importance pour la suite) :

Matrice de 2x2 boutons poussoir

Les boutons poussoirs sont repérés avec les lettres A, B, C, D. Les numéros correspondent aux numéros des pins numériques de l'Arduino.

Nous commençons au pin 2, car les pins 0 et 1 de l'Arduino peuvent être utilisés par la communication avec le moniteur série dans votre programme. C'est une bonne habitude à prendre...

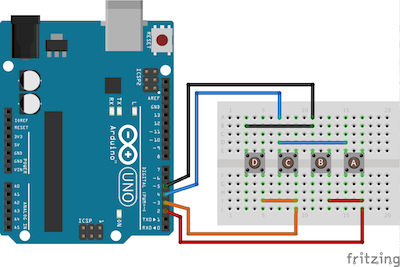
**Les contraintes techniques**

Contrairement à la LED, qui a deux pattes de connexion, le bouton poussoir, qui est pourtant un dipôle, a dans sa forme commerciale, 4 pattes de connexion. Il est de plus assez large par rapport aux trous de la breadboard.

Du coup, connecter 4 boutons poussoirs sur une bread board en gardant la forme d'une matrice carré est impossible ( à moins d'utiliser plusieurs breadboards). En effet, le bouton poussoir se connecte sur une breadboard, en le mettant à cheval par dessus la ligne centrale. Nous n'allons donc pas pouvoir connecter nos boutons en carré, mais en ligne !

Bien sûr, il est tout à fait possible de réaliser un montage soudé, mais ce n'est pas l'objectif ici.

Voici l'image de montage :

Matrice de 2x2 boutons poussoirs en ligne

Pour vous faciliter le repérage, j'ai utilisé le même code couleur entre le schéma et l'image de montage, ainsi que le même repérage des boutons.

Vous pouvez observer que :

* Les boutons A et B sont bien sur la ligne attachée au pin 2 (fil rouge) qui sera la ligne 0 ;
* Les boutons C et D sont bien sur la ligne attachée au pin 3 (fil orange) qui sera la ligne 1 ;
* Les boutons A et C sont bien sur la colonne attachée au pin 4 (fil bleu) qui sera la colonne 0 ;
* Les boutons B et D sont bien sur la colonne attachée au pin 5 (fil noir) qui sera la colonne 1 ;
* Chaque bouton a bien une coordonnée unique. Par exemple, le bouton A est à la ligne 0, colonne 0.

Pour réaliser une matrice de 3x3, le principe est le même : on repère bien les lignes et les colonnes pour que chaque bouton ait une coordonnée unique.

Il nous faut maintenant réaliser le programme qui va gérer les appuis sur les boutons.

**Programmez d'une matrice de boutons**

Nous avons vu au chapitre précédent que la programmation d'une matrice de LED se base sur une écriture de chaque coordonnée (ligne/colonne) en position haute ou basse.

La matrice de boutons procède de la même manière, mais au lieu d'écrire "(mode OUTPUT)", nous allons lire les boutons "(mode INPUT)" à chaque coordonnée.

Voici la démarche générale :

* Les pins de lignes sont en mode OUTPUT, pour envoyer le courant ;
* Les pins de colonne sont en mode INPUT, pour recevoir le courant ;
* On place un pin de ligne en état Haut ;
* On lit chaque pin de colonne ;
* On replace le pin de ligne à l'état Bas pour préparer la lecture de la ligne suivante.

Voici donc le programme :

int ligne[2]={2,3}; //tableau pour stocker les pins des lignes

int colonne[2]={4,5}; //tableau pour stocker les pins des colonnes

void setup() {

//mise en OUTPUT et LOW des lignes et INPUT des colonnes

for (int i=0;i<2;i++){

pinMode(ligne[i], OUTPUT);

digitalWrite(ligne[i],LOW);

pinMode(colonne[i], INPUT);

}

Serial.begin(9600); //communication série

}

void loop() {

test(); //appel de la fonction test

}

//fonction de test des états des boutons

void test(){

for (int l = 1; l >=0; l--) { //boucle qui parcourt le tableau des lignes

digitalWrite(ligne[l], HIGH); //on place la ligne en HIGH

for (int c = 1; c >=0; c--) { //boucle qui parcourt le tableau des colonnes

boolean etat=digitalRead(colonne[c]); //on lit le résultat

if (etat) { // si etat==1

Serial.print(" D "); //on écrit D pour down

}

else { //sinon

Serial.print(" U "); //on écrit U pour up

}

}

digitalWrite(ligne[l], LOW); //on place la ligne en LOW

}

Serial.println(); //saut de ligne

delay(100); //petite attente

}

Afin que l'affichage corresponde à la disposition des bouton sur la breadboard, vous remarquerez que nous parcourons les boucles à l'envers. Nous commençons ainsi par l'affichage de l'état du bouton D, puis C, puis B, et enfin A.

Testez le programme (il faut ouvrir le moniteur série pour voir le résultat).

Vous voyez que lorsque vous appuyez un bouton, la lettre "D" s'affiche à la bonne position. Le délai de 100 ms permet de gérer correctement la lecture.

Vous pouvez aussi tester l'appui sur plusieurs boutons à la fois !

Il y a des combinaisons qui ne fonctionnent pas ! C'est normal ?

(Oui très chère Cunégonde, et je m'en vais de ce pas en expliquer la raison...)

Prenons l'exemple d'un appui simultané sur A et C. Pour la lecture de A, la ligne 0 est en HIGH, la ligne 1 est en LOW, et la lecture se fait sur la colonne 0.

Si l'on observe le sens du courant, on voit qu'il peut circuler du pin 2 vers le pin 4, ce qui correspond à notre pin de lecture.

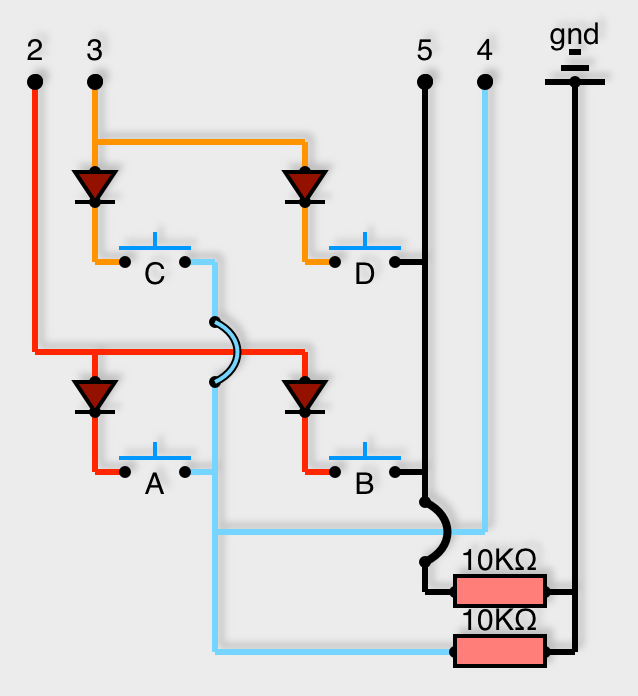
Mais comme le contact C est aussi enfoncé, il se dirige aussi vers le pin 3 (de HIGH vers LOW) ! Ce qui répartit le courant entre les deux pins et ne permet pas de réaliser une mesure d'un état haut pour le pin de lecture.

Il nous faut donc empêcher le courant de "remonter" vers le pin 3. La solution est simple, nous allons ajouter à notre montage une diode (non lumineuse) qui va bloquer le courant dans ce sens.

En fait nous allons ajouter au total 4 diodes (une par bouton), ce qui va empêcher le courant de "remonter".

Et tant que nous y sommes, pour éviter des données erratiques de lecture (voir le [chapitre sur les boutons poussoirs](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-bouton-poussoir)), nous allons ajouter une résistance (10 kΩ) à chaque colonne qui sera reliée au ground. Ainsi la lecture de l'état sera nette (HIGH ou LOW).

Voici donc le montage final à réaliser pour pouvoir afficher les lettres des boutons poussoirs actionnés à la manière d’un clavier à appuis multiples.

Matrice de 2x2 boutons poussoir, avec diodes et résistance pull-down

Si votre montage est correct, vous avez maintenant accès à toutes les combinaisons (soit 2^4=16 possibilités). En effet, chacun des 4 boutons peut avoir 2 états possibles (haut ou bas).

* Nous avons utilisé une matrice 2x2 pour l'exemple. Elle n'a pas d'intérêt réel, sauf pour le côté théorique. En effet, elle utilise 4 pins pour fonctionner, ce qui reviendrait à connecter les 4 boutons sur 4 pins séparés. Comme pour les matrices de LED, à partir de 3x3, ce montage devient intéressant pour économiser les pins.
* Je vous ai présenté cette méthode de matrice de boutons pour parler de la possibilité de ne pas être connecté à un ordinateur, alors que j'utilise le moniteur pour réaliser des affichages (qui servaient de contrôle). Là encore, c’était pour simplifier les explications de départ, mais une fois que vous avez bien compris comment réaliser une matrice de boutons fonctionnelle, vous pouvez tout à fait l'utiliser pour piloter votre carte sans connexion à un ordinateur ! Par exemple pour créer une télécommande, un clavier de codage, un lanceur de fonctions précises...
* En remplaçant les boutons poussoir par des micro-rupteurs, vous avez aussi la possibilité, grâce à ces matrices,  de tester des contacts multiples sur un robot. ;)

Et si nous réalisions un petit exercice ?

**TP : Réalisez un coffre-fort virtuel**

Je vous propose de réaliser un programme qui utilise la matrice de boutons poussoir pour saisir un code qui ouvrirait un "coffre fort virtuel". L'idée est la suivante :

* Une combinaison est stockée dans une variable dans le programme.
* On appuie sur les boutons pour former une combinaison et on relâche.
* Si la combinaison est la bonne une LED  s'allume, sinon elle reste éteinte.
* Au bout de trois essais, la LED se met à clignoter rapidement pour indiquer l'auto-destruction du coffre-fort.
* On appuie sur le bouton "reset" situé sur la carte Arduino pour relancer le programme.

Il vous faut donc prévoir pour réaliser le montage :

* La matrice de boutons poussoir que l'on vient de réaliser dans ce chapitre ;
* Une LED connectée avec la résistance qui va bien.

Du côté du programme :

* Une variable pour le code (je vous conseille un type chaîne de caractère, exemple : "DUUD" avec D=down et U=Up) ;
* Un moyen de lire et stocker la combinaison saisie (pour la comparer) ;
* Une variable qui compte les essais ;
* Une solution pour faire clignoter la diode au bout de 3 essais (sûrement avec la gestion du temps, donc l'utilisation de la fonction  millis()) ;
* Quelques fonctions pour la lisibilité du programme (fonction de comparaison, de lecture de matrice, de clignotement... par exemple).

Je vous laisse réfléchir. Prenez votre temps, et là encore, essayez de trouver une solution par vous-mêmes avant de passer à la correction. ;)

**TP : Correction**

Voici la solution que je vous propose pour le code (le montage ne devrait pas vous poser de problème) :

/\*

\* Coffre-Fort by Nanomaitre 2015

\* LED connectée au pin 13 avec résitance 220Ω

\* matrice de boutons poussoirs 2x2 :

\* lignes : pins 2 et 3

\* colonnes : pins 4 et 5

\*/

char code[5] = "DUUD"; //chaine de caractère pour stocker la combinaison

char codeSaisi[5] = "UUUU";//chaine de caractère pour stocker la saisie

char codeActuel[5] = "UUUU"; // tableau de char pour stocker l'état des boutons

int essai = 3; // variable pour stocker le nombre d'essais

boolean ok = 0; // boolean pour stocker si code trouvé ou non

boolean etatLED = 0; // boolean pour gérer le clignotement

int pinLED = 13; // variable pour stocker le pin de la LED

int ligne[2] = {2, 3}; //tableau pour stocker les pins des lignes

int colonne[2] = {4, 5}; //tableau pour stocker les pins des colonnes

unsigned long tpsDep = millis(); //initialisation du temps de départ pour clignotement

void setup() {

//mise en OUTPUT et LOW des lignes et INPUT des colonnes

for (int i = 0; i < 2; i++) {

pinMode(ligne[i], OUTPUT);

digitalWrite(ligne[i], LOW);

pinMode(colonne[i], INPUT);

}

pinMode(pinLED, OUTPUT); //mise du pin de LED en mode OUTPUT

}

void loop() {

if (!ok && essai > 0) { // si code non trouvé et essai restants

test(); // appel de la fonction de lecture de matrice

if (!relache()) { // si un appui est lu

while (!relache()) { // on répète tant qu'un bouton au moins est appuyé

test(); //lecture de la matrice

}

compare(); //au relâchement, on compare avec la combinaison

}

}

else if (ok) { // sinon, teste si la combinaison est trouvée

digitalWrite(pinLED, HIGH); //on allume la LED

}

else { //sinon (nombre d'essai à 0

unsigned long tpsAct = millis(); //temps actuel

if (tpsAct - tpsDep > 100) { //si différence > 100 ms

etatLED = !etatLED; //on inverse l'état de la LED

digitalWrite(pinLED, etatLED); //affichage de l'état de la LED

tpsDep = tpsAct; // réinitialisatin du temps de départ.

}

}

}

//fonction de test des état des boutons de la matrice

void test() {

int pos = 0; //position dans le tableau de char

for (int l = 1; l >= 0; l--) { //boucle qui parcourt le tableau des lignes

digitalWrite(ligne[l], HIGH); //on place la ligne en HIGH

for (int c = 1; c >= 0; c--) { //boucle qui parcourt le tableau des colonnes

boolean etat = digitalRead(colonne[c]); //on lit le résultat

if (etat) { // si etat==1

codeActuel[pos] = 'D'; //on écrit D dans le tableau de char

}

else { //sinon

codeActuel[pos] = 'U'; //on écrit U dans le tableau de char

}

pos++; //on passe à la position suivante

}

digitalWrite(ligne[l], LOW); //on place la ligne en LOW

}

if (!relache()) { //on teste si l'appui est toujours actif

// si oui on met à jour la position saisie

for (int t = 0; t < 4; t++)

codeSaisi[t] = codeActuel[t];

}

delay(100); //petite attente

}

//fonction de comparaison entre position saisie et combinaison

void compare() {

for (int i = 0; i < 4; i++) { // on parcours les deux tableaux

if (code[i] != codeSaisi[i]) { // si différence

essai--; // on enlève un essai

return; //on sort du test

}

}

//si on arrive là c'est que le code correspond

ok = 1; //donc on ouvre le coffre

}

//fonction de test si tout relaché

boolean relache() {

//on parcourt l'état de la matrice

for (int i = 0; i < 4; i++) {

if (codeActuel[i] == 'D') //si une touche est enfoncée

return 0; //on renvoie 0

}

//sinon

return 1;//on renvoie 1

}

Ce code n'est pas forcément le seul possible pour réaliser ce TP. Si vous avez réfléchi au problème, et testé vos solutions, vous avez dû remarquer que la difficulté majeur se situe dans le test d'appui et la comparaison.

En effet, l'appui des boutons est lu très rapidement, et il faut être sûr du moment où l'on compare. J'ai créé pour ce faire la fonction relâche() qui teste si la matrice ne contient aucun appui. Elle me permet de temporiser l'appui (utilisation de la condition while()) de ne récupérer que le code saisi (et non un code vide au moment du relâchement) et de tester ce code récupéré, après relâchement.

Une petite astuce vous permet de faire fonctionner deux diodes en opposition : vous connectez la première sur le pin de commande (patte +) en la reliant au ground (patte -), et la deuxième, du même pin de commande (patte -) en la reliant au +5V (patte +). Le tout bien sûr avec les bonnes résistances . Du coup, lorsque le pin de commande est à LOW, la LED reliée au 5V s'allume et l'autre est éteinte, et lorsque le pin est à HIGH, la LED reliée au ground s'allume et l'autre s'éteint ;)

En utilisant une LED verte et une rouge, l'effet est très lisible.

Je pense que nous avons fait le tour des matrices de boutons, voyons maintenant les boutons en série...

**Les boutons en série**

Je vais aborder avec vous une autre façon de récupérer les informations de plusieurs boutons tout en économisant des pins de la carte Arduino.

Le titre "les boutons en série", n'est pas tout à fait juste. En effet des boutons en série (les uns à la suite des autres) ne serviraient pas à grand-chose, car il faudrait que tous soient appuyés en même temps pour que le courant passe.

Il s'agit plutôt de faire un montage qui va, en fonction du bouton appuyé, faire varier la tension.

Mais un bouton est lu à l'état haut ou bas ? Ce n'est pas une différence de tension ?!

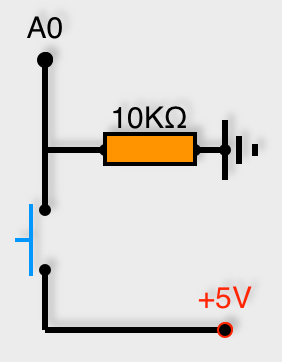
(Cunégonde, je me demande parfois pourquoi vous suivez ce cours... on dirait que vous savez déjà tout !)

Nous n'allons pas utiliser les pins numériques pour lire nos boutons, mais les pins analogiques – un seul pin analogique pour être précis.

Je vous rappelle que les pins analogiques de l'Arduino, sont des convertisseurs analogiques/numériques (CAN) qui peuvent convertir une tension entre 0V et 5V en nombre entre 0 et 1023. Chaque unité représentant environ 5 mV.

Il nous faut donc trouver un moyen de lire des valeurs différentes en fonction du bouton appuyé.

Nous avons vu dans le cours d'initiation comment connecter un bouton poussoir avec une résistance pull-down. Nous utilisions un pin numérique pour lire l’état du bouton poussoir. Je vous propose de faire le montage suivant, c'est le même mais en se connectant sur un pin analogique (le A0) :

Connexion d'un bouton poussoir avec résistance pull-down sur pin analogique

Vous pouvez afficher les valeurs lues à l'aide du programme suivant :

void setup() {

Serial.begin(9600);//communication série

}

void loop() {

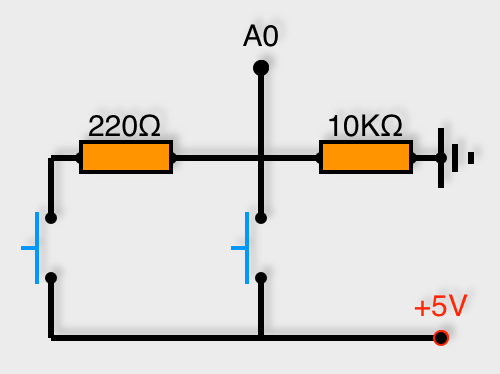
Serial.println(analogRead(A0)); //affichage de la valeur lue par le CAN 0

delay(10); //petite attente

}

Lorsqu'on appuie sur le bouton, on lit la valeur 1023, sinon la valeur 0. Ce qui correspond aux deux états low (0V) et high (+5V) transformées en valeurs numériques par le CAN.

Et bien l'idée est d'ajouter un bouton en série avec une résistance (220Ω pour l'exemple) qui viendrait se connecter au pin A0. Voici le montage :

Deux boutons poussoir en dérivation (dont l'un avec une résistance en série)

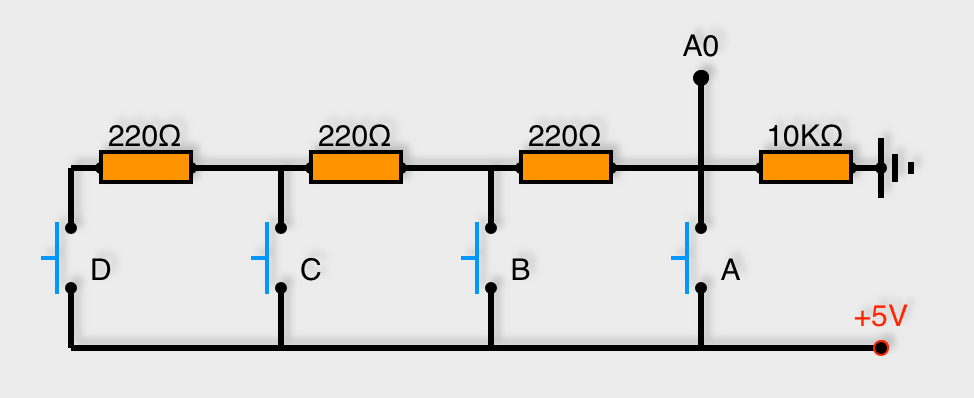
Si on appuie sur le bouton de droite, le +5V est relié au pin A0, le CAN indique donc 1023 (car il reçoit du +5V)

Mais, si l'on appuie sur le bouton de gauche, le courant passe à travers la résistance (220Ω) et se dirige vers le pin A0.

Le CAN va donc lire une tension inférieure à +5V et nous renverra un nombre inférieur à 1023.

Faites le test, il vous suffit d'utiliser le même programme que précédemment.

Je vous propose maintenant, en suivant le même principe  et de connecter 4 boutons. Voici le montage :

4 boutons poussoir montés en dérivation

Que se passe-t-il si on appuie sur D ? Et bien le courant passe par D puis par les 3 résistances (qui du coup sont en série) avant d'aller rejoindre A0. La valeur de la tension sera donc encore amoindrie.

Hé, mais si j'appuie sur plusieurs boutons, la valeur reste la même !

(Quel esprit scientifique Cunégonde, toujours tester, toujours chercher !)

En effet, c'est la limite de ce montage. On peut arriver à repérer différents boutons, mais, contrairement au réseau de bouton vu plus haut, ce montage ne permet pas la lecture de plusieurs boutons appuyés simultanément.

**Le programme pour les boutons en série**

Je dirais qu'il n'y a qu'une difficulté et elle est matérielle. En effet, les résistances ont une marge d'erreur de 5% (bague or sur la résistance). Ce qui fait qu'on ne peut pas calculer avec certitude q'un nombre correspond à une valeur.

Dans notre cas, les valeurs affichées (à par celles du bouton A) oscillent. Nous allons donc devoir tester des intervalles de valeurs pour récupérer le bouton appuyé.

Voici un premier programme :

void setup() {

Serial.begin(9600);//communication série

}

void loop() {

int valeur=analogRead(A0);

Serial.print(valeur); //affichage de la valeur lue par le CAN 0

if (valeur>=1023){

Serial.println(" bouton A");

}

else if (valeur>=999){

Serial.println(" bouton B");

}

else if (valeur>=979){

Serial.println(" bouton C");

}

else if (valeur>=959){

Serial.println(" bouton D");

}

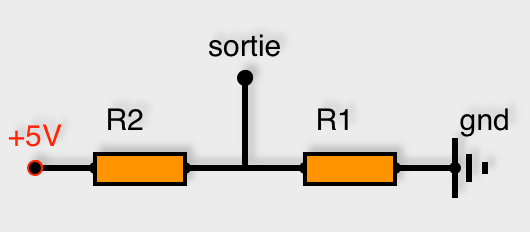
delay(10); //petite attente

}

J'ai volontairement fait les tests en utilisant la valeur minimale lue (en utilisant le programme plus haut) et en enlevant 1. Il m'a donc fallu noter les valeurs pour chaque appui.

Le montage qu'on réalise lorsqu'on met ces boutons en dérivations avec une résistance en série, s'appelle un diviseur de tension. D'ailleurs, le montage avec une résistance pull-down est déjà un montage en diviseur de tension.

Le principe est le suivant. Voici un montage :

Schéma pour pont diviseur de tension

La première formule  à connaître est la suivante :

tension de sortie= (r1/(r1+r2)) x tension d'entrée

Dans notre cas, r1 vaut 10000Ω, r2 vaut (pour le bouton B) 220Ω, la tension d'entrée est de 5V, on obtient donc :

sortie=(10000/(10000+220))\*5=4,8923 V

Si on mappe cette valeur entre 0 et 1023, on obtient 1000 (valeur qui correspond à peu près à celle lue par le CAN)

On peut donc calculer les intervalles de valeur (en prenant en compte +-5% pour les résistances à bande dorées) pour obtenir la fourchette de résultats.

Si plusieurs résistances sont en série, on ajoute leur valeur (pour le bouton C, r2 vaut donc 440Ω +-5%).

Ce calcul peut nous permettre de savoir à l'avance les paliers en fonction des résistances (attention, il faut utiliser des résistances de valeur identiques !) :

Voici un programme qui permet d'afficher les valeurs en fonction du nombre de boutons :

float r1=10000.0; // valeur pour R1

float r2=220.0; // valeur pour R2

float p=5.0; // pourcentage d'erreur

float ue=5000.0; // tension d'entrée en mV

int nbBoutons=4; // nombre de boutons

void setup() {

Serial.begin(9600);//communication série

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*");

for (int t=0;t<nbBoutons;t++){

float pr2=r2\*p/100; //valeur du pourcentage

float usmin=(r1/(r1+t\*(r2+pr2)))\*ue; // valeur sortie min

float usmax=(r1/(r1+t\*(r2-pr2)))\*ue; // valeur sortie max

float us=(r1/(r1+t\*(r2)))\*ue; // valeur sortie théorique

//affichage

Serial.print ("Bouton ");

Serial.print (t);

Serial.print (" : ");

Serial.print (map (int(usmin),0,5000,0,1023));

Serial.print (" < ");

Serial.print (map (int(us),0,5000,0,1023));

Serial.print (" < ");

Serial.println (map (int(usmax),0,5000,0,1023));

}

}

void loop() {

}

Je vous fais remarquer au passage l'utilisation du type float. C'est pour utiliser les nombres à virgules.

Il est important pour qu'un nombre entier soit considéré comme nombre à virgule, de le définir avec sa virgule. Exemple 3 doit être défini par 3.0

En utilisant les valeurs minimum de chaque palier, on peut stocker dans un tableau les valeurs à utiliser sans avoir à faire de relevés.

Voici un programme qui nous dit quel bouton est appuyé grâce à un tableau calculé à l'avance :

float r1=10000.0; // valeur pour R1

float r2=220.0; // valeur pour R2

float p=5.0; // pourcentage d'erreur

float ue=5000.0; // tension d'entrée en mV

const int nbBoutons=4; // nombre de boutons

int valeurs[nbBoutons]; // tableau de stockage des valeurs basses

void setup() {

Serial.begin(9600);//communication série

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*\*");

for (int t=0;t<nbBoutons;t++){

float pr2=r2\*p/100; //valeur du pourcentage

float usmin=(r1/(r1+t\*(r2+pr2)))\*ue; // valeur sortie min

valeurs[t]=map (int(usmin),0,5000,0,1023); //ajout d'une valeur

}

}

void loop() {

int val=analogRead(A0); // lecture du CAN 0

if (val){ // si différent de 0

for (int t=0;t<nbBoutons;t++){ // parcours des valeurs

if (val>=valeurs[t]){ //test si supérieur à valeur concernée

// affichage

Serial.print ("Bouton ");

Serial.print (t);

Serial.println (" down");

break; //sortie de boucle

}

}

}

}

On peut, typiquement, grâce à cette méthode, utiliser des claviers construits de façon personnelle en n'utilisant qu'un seul pin de l'Arduino !

En effet, en utilisant des résistances de 220Ω, on peut atteindre théoriquement une utilisation de 50 boutons ! https://openclassrooms.com/bundles/common/images/smiley/waw.png

Sans pouvoir, bien sûr appuyer plusieurs boutons simultanément.

À vous de construire votre propre calculatrice, ou votre clavier de saisie (ou votre clavier utile à vos seules fins) ou de pouvoir tester plusieurs contacts de type micro-rupteurs.

Je pense que nous pouvons nous arrêter là pour le moment.

**En résumé**

Vous avez vu dans ce chapitre, comment créer un clavier pour votre Arduino grâce à deux méthodes :

* La matrice de boutons : qui permet de tester l'appui simultané de plusieurs bouton, mais qui reste gourmande en pin
* La série de boutons : qui permet de savoir quel bouton est appuyé parmi un nombre assez grand de boutons, mais qui ne gère pas les appuis simultanés.

Comme pour les matrices de LED, il est possible d'utiliser des puces de gestions d'entrée/sortie qui fonctionnent en I2C pour gérer les matrices de boutons.

Nous allons dans le chapitre suivant étudier une interface d'affichage proposée dans le commerce...

**Programmez un écran LCD**

Créer des interfaces avec des matrices de LED vous paraît fastidieux ? Ça tombe bien, il existe dans le commerce des afficheurs LCD ou LED déjà préconçus pour fournir une interface de lecture simplifiée et qui permettent tout un tas d’usages.

Nous allons voir dans ce chapitre comment utiliser l'un de ces afficheurs : un écran LCD 16 caractères sur deux lignes. Comme nous n'allons pas réinventer ce qui existe déjà, nous allons surtout apprendre à utiliser la bibliothèque qui est liée à ce matériel. Vous concevrez aussi quelques programmes pour avoir une idée des possibilités de ces afficheurs, qui permettent de lire des informations, même si l'Arduino n'est plus connecté à votre ordinateur.

L'utilisation de l'écran se fera directement avec les pins de l'Arduino et non par protocole I2C. Il est possible de piloter l'écran LCD par I2C grâce à des puces spécialisées (PCF8574 par exemple), mais ce ne sera pas l'objet de ce chapitre.

Pour suivre ce chapitre, il vous faut prévoir un  LCD compatible avec le pilote Hitachi HDD44780 (avec ou sans rétro éclairage) dont la description est donnée plus loin.

Alles, c'est parti !

**L'écran LCD**

Les écrans LCD existent depuis 1971. Ils n'ont pas cessé de se développer depuis, et équipent maintenant bien des appareils à affichage embarqué (appareils photo, digicodes, montres, téléphones...).

LCD est l'abbréviation anglaise de "liquid crystal display" qui veut dire : afficheur à cristaux liquides. Je ne vais pas entrer dans le détail du fonctionnement de tels afficheurs. Il faut juste savoir que cette technologie permet de créer des écrans plats qui consomment peu d’énergie.

L'écran LCD que je vous propose d'acquérir (ou en tous cas d'étudier) est un écran permettant l'affichage de 16x2 caractères, c'est-à-dire deux lignes de 16 caractères.

Le coût de ce type d'écran varie entre 7€ et 15€ (tout dépend d'où vous le commandez et des fonctionnalités dont il dispose, entre autre le rétro-éclairage).

Il en existe de plusieurs formes. Voici celui que j'utilise :

Afficheur LCD 16x2, affichage blanc sur fond bleu (www.mchobby.be)

Pourquoi celui-ci ?

Pour une question de goût personnel. En effet il en existe d'autres avec des couleurs d'affichage différentes. À vous de choisir. Cet écran est de plus rétro-éclairé, ce qui permet de lire de jour comme de nuit.

L'important c'est que vous choisissiez un écran compatible avec la bibliothèque fournie par l'Arduino pour les écrans à cristaux liquides. On repère ce type d'écran LCD car il dispose de 16 trous (où peuvent être soudées des broches de connexion). Voici un autre exemple d'écran LCD :

Afficheur LCD 16x2, affichage noir sur fond vert, non rétro-éclairé (www.arduino.cc)

On voit bien en haut de l'afficheur, 16 marques de soudure.

C’est une bonne façon de repérer les afficheurs compatibles avec la bibliothèque LCD. Ces afficheurs peuvent communiquer avec l’Arduino via cette bibliothèque grâce au pilote Hitachi HDD44780.  Il peut être intéressant de lire cette [présentation du HD44780](https://fr.wikipedia.org/wiki/HD44780) car elle renseigne correctement sur bien des questions qu'on peut se poser dans l'utilisation (et la connexion) de ce matériel.

Attention, comme dit précédemment, je vais parler de la programmation d’afficheurs directement avec l’Arduino, et non via le protocole I2C. Il existe des afficheurs LCD munis d’une puce I2C qui permet de communiquer via ce protocole avec une autre bibliothèques, mais je n’en parlerai pas ici.

Pour la connexion, je vais vous en parler juste après.

Il faut savoir qu'il y a deux façons de piloter l'écran :

* soit en utilisant les 16 connexions (mode 8 bits),
* soit en utilisant 4 connexions de moins (mode 4 bits), ce qui économise des pins de l'Arduino.

Bon, je ne peux plus reculer. Je suis obligé de vous faire un point (léger) sur les notions de nombre décimal, binaire et  hexadécimal...

**Décimal, binaire et hexadécimal**

Le **système décimal** est en base 10. Ce qui veut dire qu'on passe à la dizaine supérieure toutes les dix unités (ou bien on augmente le nombre des milliers toutes les 10 centaines...).

Une image simple est d'imaginer un compteur de voiture (les anciens, ceux qui tournent). On voit bien qu'à chaque tour (donc 10 unités) des centaines de mètres, on fait tourner d'un cran les kilomètres. Ce système nous est familier car on l'utilise tout le temps. Voici les 16 premiers nombres du système décimal :

0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16.

Le **système binaire** est en base 2, c'est-à-dire que chaque fois qu'on passe deux unités (0 puis 1). Comme on ne peut pas parler de dizaine (puisque c'est en base 10) on va parler de bit. Le bit en position 0 est celui le plus à droite du nombre, puis le bit en position 1, puis 2... Donc pour augmenter les bits, on ajoute 1 au bit de gauche et on remet le bit de droite à 0. Voici donc les 16 premiers nombres du système binaire :

0000,0001,0010,0011,0100,0101,0110,0111,1000,1001,1010,1011,1100,1101,1110,1111.

Prenez votre temps pour comprendre le principe d'augmentation du nombre.

Enfin le **système hexadécimal**est en base 16. Donc sa dizaine ne change que toutes les 16 unités. Comme nos chiffres s'arrêtent à 9, il nous a fallu inventer des chiffres après 9. Les informaticiens ont donc utilisé des lettres. Du coup, voici les 16 premiers nombres en hexadécimal :

0,1,2,3,4,5,6,7,8,9,*A*,*B*,*C*,*D*,*E*,*F*.

Ces trois systèmes de numération sont utilisés en informatique. Voici le même nombre écrit en décimal, binaire et hexadécimal : 213 (décimal), 1101 0101 (binaire), D5 (hexadécimal).

Vous pouvez assez facilement passer de l'une à l'autre des numérations par un calcul mathématique. Mais, vous pouvez faire ça encore plus simplement avec des outils dédiés (calculatrice en mode programmeur sur votre ordinateur, site de conversions, etc.) ou même avec un petit programme sur votre Arduino !

 En voici un exemple :

void setup() {

Serial.begin(9600); // communication série

int nombre=220; // nombre à convertir

Serial.println("\*\*\*\*\*\*\*\*\*\*\*\*");

Serial.print ("Dec : ");

Serial.println (nombre,DEC); //converti en décimal

Serial.print("BIN : ");

Serial.println(nombre,BIN); // converti en binaire

Serial.print("HEX : ");

Serial.println(nombre,HEX); // converti en hexadécimal

}

void loop() {

}

Ce programme affichera le nombre 220 en décimal, binaire et hexadécimal.

* Si vous voulez placer un nombre binaire dans une variable, ajoutez un B majuscule avant la suite de 1 et 0 de ce nombre.  *Exemple :* int nombre=B10010  le B signifie que votre nombre est écrit en binaire.
* Pour enregistrer un nombre en hexadécimal, il faut ajouter "0x" (zéro et x minuscule) devant ce nombre.  
  *Exemple* : int nombre=0xFA12F  le "Ox" signifie que votre nombre est en hexadécimal.

Il existe un quatrième type de numération. La numération octale (en base 8). Il faut mettre un 0 (zéro) avant le nombre saisi. Pour que l'Arduino affiche un nombre en base 8, il suffit d'écrire  Serial.println(nombre,OCT);  OCT signifie octal. Je n'utilise pour ma part jamais cette numération. Vous pouvez avoir quelques renseignements complémentaires sur [ce wiki](https://fr.wikipedia.org/wiki/Syst%C3%A8me_octal).

Quel est l'intérêt de ces trois systèmes ?

* Le système décimal est celui qu'on utilise tous les jours, il est lisible facilement nous les humains ^^.
* Le système binaire est le celui qu'utilise la machine. Il est facile de construire un nombre en allumant ou éteignant des connexions. En effet, avec 4 LED côte à côte, vous pouvez créer (en les allumant pour 1 et en les éteignant pour 0) les nombres entre 0 et 16. Par exemple : allumé, éteint, éteint, allumé correspond à 1001 soit 9 en décimal.
* Le système hexadécimal, permet de coder les nombres de 8 bits (donc 16 positions binaires) sur seulement 2 caractères. En effet, le nombre binaire 1111 1111 correspond à FF en hexadécimal (255 en décimal). C'est donc un gain de place pour le codage

Quel rapport avec notre afficheur ?

Et bien c'est ce dont je parlais, soit on utilise 8 broches (plus les autres pour l'alimentation) pour coder l'information (donc 16 pins qui se mettront à 1 ou 0 pour former un nombre de 8 bits) ou seulement 4 broches (plus celles de l'alimentation) mais dans ce cas le nombre 8 bits sera séparé en deux :

* Les quatre premiers bits seront envoyés. On appelle ces bits les bits de poids fort (les quatre plus à gauche).
* Puis les 4 seconds. On appelle ces bit les bits de **poids faible** (les 4 plus à droite).

Cela diminue donc par deux la vitesse de communication.

Il faut savoir que la communication avec ces afficheurs, et les possibilités de l'Arduino, réduisent l'unité d'information à 8 bits (ou deux fois 4 bits). Les ordinateurs actuels peuvent communiquer à l'aide d'unités d'information de 32 bits et 64 bits. La vitesse de communication est donc bien plus rapide. Tout dépend donc du matériel.

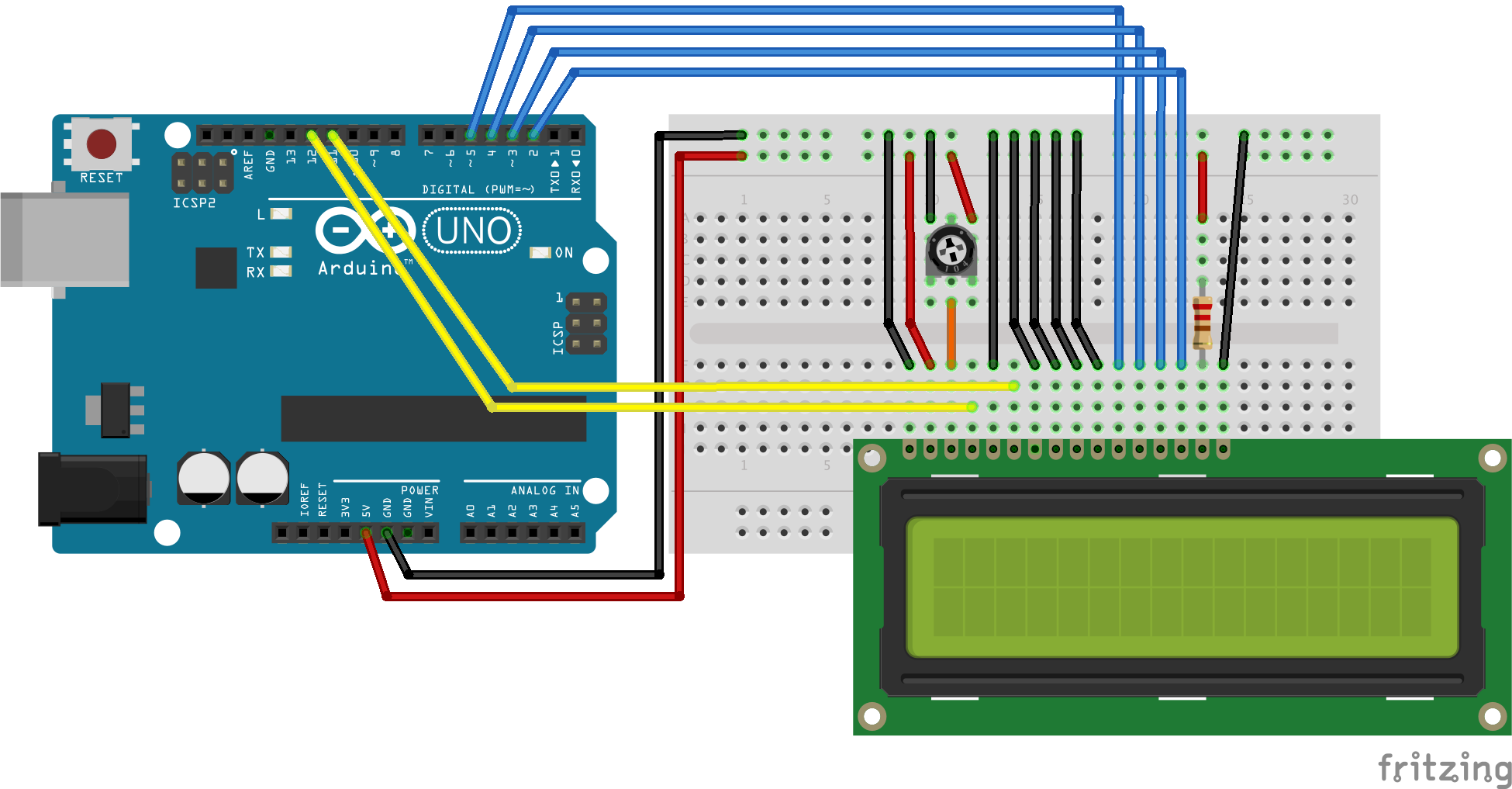
Rassurez-vous, nous n'allons pas coder (même si c'est tout à fait possible) nous-mêmes la communication entre l'Arduino et l'afficheur LCD !

Si vous avez jeté un œil aux explications sur le protocole de communication du HD44780  (qui sont bien faites mais pas forcément facilement compréhensibles), vous avez vu que pour échanger des informations, on utilise un protocole précis qui permet à chaque partie (Arduino et écran LCD) de se comprendre. C'est à cause de ce protocole que nous devons utiliser plus que 8 pins (ou 4 pins) pour connecter l'écran LCD. En effet, il nous faut aussi des pins pour l'adresse, le timing de l'échange d'info, l'alimentation de l'écran... Regardons de plus près comment connecter un écran LCD à l’Arduino en mode 4 bits !

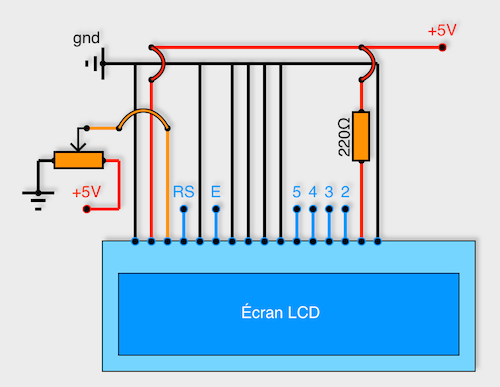
**Connectez l’écran LCD à l’Arduino en mode 4 bits**

Comme l'Arduino (malgré ses 16 MHz) n'est pas une bête de course, nous n'utiliserons que le mode 4 bits. Même si la communication est deux fois plus lente, ce type d'afficheur ne sera de toute façon pas utilisé pour des changements d'information rapides. Le mode 4 bits est donc largement suffisant, plus simple et moins gourmand en pins.

Voici une image pour la connexion des différentes broches :

Connexion d'un écran LCD avec l'Arduino

Alors ça peut paraître un peu ardu comme ça, mais la connexion respecte une certaine logique (heureusement ;) ). Voici un schéma que j'ai tenté de simplifier :

Connexion écran LCD avec Arduino.

 En partant de la gauche, voici à quoi servent les pins :

* Les deux premiers pins tout à gauche servent à l'alimentation de l'écran.
* Le troisième pin est connecté à un potentiomètre et sert pour régler l'affichage (le contraste de l'écran).
* Le quatrième, noté RS, est connecté au pin 12 de l'Arduino dans notre exemple. Il sert à sélectionner la zone mémoire de l'écran LCD dans laquelle nous allons écrire (Register Select).
* Le cinquième doit toujours être connecté au ground. C'est un sélecteur de mode lecture ou écriture. On peut le connecter à un pin, mais dans notre cas c'est inutile. Comme il doit recevoir un signal à 0V, on le connecte au ground (état R/W).
* Le sixième, noté E, est connecté au pin 11 de l'Arduino dans notre exemple. Il permet de lancer ou non l'écriture dans les zones mémoires (Enable).
* Les quatre suivants (reliés au ground) servent pour la communication 8 bits. Pour la communication 4 bits, il est conseillé de les relier au ground. Ils représentent les bits de poids fort.
* Les quatre qui suivent,  notés 2, 3, 4, 5, se connectent dans notre exemple sur les pins 2, 3, 4, 5 de l'Arduino.  Ils servent pour la communication (8 bits ou 4 bits) et doivent toujours être connectés. Ils représentent les bits de poids faible (ou servent pour envoyer d'abord les bits de poids faible, puis les bits de poids fort)
* Les deux pins tout à droite servent pour alimenter la LED du rétro-éclairage.

En mode 4 bits, il est finalement important de bien repérer les 6 pins en bleu sur le schéma.

Je vous laisse brancher votre LCD en suivant ce modèle et ensuite mettre sous tension l'Arduino...

Si vous tournez le potentiomètre, vous voyez le contraste changer.

Voyons maintenant comment écrire sur votre écran LCD...

**Écrivez sur un écran LCD**

Nous allons procéder en deux étapes. Tout d'abord ce qui concerne la bibliothèque, puis les méthodes d'écritures elle-même.

**Chargez la bibliothèques pour écran LCD**

Alors pour charger et installer la bibliothèque qui permet une communication directe entre l'Arduino et l'écran LCD, il suffit de suivre la procédure maintenant presque habituelle :

Croquis -> Include Library -> Manage Librairies

Dans l'onglet de recherche, tapez : "LiquidCrystal". Si la bibliothèque est déjà installée, vous verrez noté "INSTALLED" à côté de son nom.

Ne chargez pas la bibliothèque pour I2C, elle ne vous sera pas utile ici...

Une fois la bibliothèque LiquidCrystal chargée, vous pouvez fermer et relancer votre IDE.

Comme la dernière fois, si la bibliothèque est bien présente, dans Fichier -> Exemples, vous devriez voir un menu LiquidCrystal.

Voyons maintenant comment utiliser cette bibliothèque.

**Programmez votre écran LCD**

Nous commençons notre programme en incluant la bibliothèque :

#include <LiquidCrystal.h>

Comme pour toute utilisation de bibliothèque,  nous allons commencer par définir un objet avec quelques paramètres :

LiquidCrystal monEcran(12,11,5,4,3,2);

En fait la forme est de type :

LiquidCrystal nomDeVariable(RS, E, d4, d5, d6, d7);

où RS correspond au numéro du pin RS, E au numéro du pin E, et les autres paramètres aux numéros des pins pour la communication 4 bits.

J’ai nommé mon objet  monEcran , vous pouvez choisir le nom que vous souhaitez bien sûr. Il existe d’autres façons d’initialiser un objet écran. Vous les trouverez dans la [documentation sur la bibliothèque](https://www.arduino.cc/en/Reference/LiquidCrystal) pour écrans LCD.

Ces deux commandes (inclusion de la bibliothèque et création de l’objet écran) se situent avant le setup().

Ensuite (un peu comme avec la bibliothèque Serial), nous devons initialiser la communication. Comme pour toute utilisation de fonctions d'objet, on met le nom de l'objet, un point, et le nom de la fonction avec entre parenthèses les paramètres éventuellement attendus. Voici l’initialisation de notre objet écran :

monEcran.begin(16,2);

Cela signifie : j'initialise la communication entre l'Arduino et l'objet monEcran avec une matrice de 16 colonnes et 2 lignes.

À partir de ce moment, vous pouvez utiliser toutes les autres fonctions de la bibliothèque !

Voici un programme simple qui affiche "Bonjour, OPENCLASSROOMS" sur deux lignes :

#include <LiquidCrystal.h> // on importe la bibliothèque

LiquidCrystal monEcran(12,11,5,4,3,2); // on crée l'objet écran

void setup() {

monEcran.begin(16,2); //on initialise la communication avec 16 colonnes et deux lignes

monEcran.clear(); // on efface l'écran

monEcran.print("Bonjour,"); // on écrit Bonjour

monEcran.setCursor(0,1); //on passe à la ligne suivante

monEcran.print("OPENCLASSROOMS"); // on finit d'écrire

}

void loop() {

}

Vous remarquerez que le retour à la ligne ne se fait pas automatiquement, il faut le prévoir.

La liste de toutes les commandes possibles sont sur la page de [documentation](https://www.arduino.cc/en/Reference/LiquidCrystal) liée à la bibliothèque.

Je vous propose pour vous entraîner, de réaliser un programme simple à base de menu...

**TP : Programmez un menu simple**

Dans ce TP, je vous propose d’utiliser l’afficheur LCD pour naviguer dans un menu, grâce à deux boutons.

Pour le montage, vous aurez besoin de :

* Un écran LCD connecté comme nous venons de le voir ;
* Deux boutons poussoir (avec résistance en pulldown) pour gérer les déplacements dans le menu, attachés aux pins 8 et 9.

Pour le programme associé, vous allez créer un menu simple structuré de la façon suivante :

* Option 1 : **LED 13**,composée de 3 sous-options - "OFF", "ON" et "Clignote". "OFF" permet d'éteindre la LED, "ON" permet de l'allumer, et "Clignote" permet de la faire clignoter. Je parle ici de la LED attachée au pin analogique 13 (voir début du cours d'initiation à l'Arduino)
* Option 2 : **Analogique**, composé de 5 sous-options - "A0", "A1", "A2", "A3", "A4" qui affichent les valeurs de CAN de 0 à 4.

Pour passer de l'option LED13 à l'option Analogique, vous utiliserez le premier bouton ; et pour naviguer entre les sous-options de chacune de ces deux options, vous utiliserez le deuxième bouton.

Le passage d'un item à l'autre affiche en première ligne le titre de l'item et en seconde ligne le sous item (soit l'état de la LED 13, soit le nom du CAN et sa valeur).

(Cunégonde et Lukas, vous travaillerez en groupe, on vous appellera le "LCD" pour Lukas Cunégonde Display !)

**TP : "Programmez un menu simple" - Correction**

Alors vous y êtes arrivé sans problème ? Vous avez même créé un menu de 10 options tellement ça vous paraissait simple ? ;)

Pour le montage, si vous avez rencontré des problèmes pour connecter les boutons poussoir avec résistance pull down, je vous invite à revoir [le chapitre sur les boutons poussoir](https://openclassrooms.com/courses/programmez-vos-premiers-montages-avec-arduino/le-bouton-poussoir) dans mon cours d'initiation à l'Arduino.

Pour le programme , voici un exemple de solution…

/\*

\* Menu avec écran LCD

\* Ce programme propose la navigation dans un menu simple

\* à l'aide de deux boutons (sur pin 8 et 9)

\* et d'un écran LCD compatible HD44780

\* Deux items : LED 13 et Lecture des CAN

\* Le menu LED 13 allume, éteint, ou fait clignoter la LED 13

\* Le menu Lecture des CAN, lit les valeurs des CAN 0,1,2 et 3 successivement

\*

\* Nanomaitre 2015 pour OpenClassRooms

\* Ce programme est dans le domaine public

\*/

#include <LiquidCrystal.h> // on importe la bibliothèque pour LCD

LiquidCrystal monEcran(12, 11, 5, 4, 3, 2); // on crée un objet LCD nommé monEcran en mode 4 bits

const int bout1 = 8; //constante du pin bouton 1

const int bout2 = 9; //constante du pin bouton 2

int posMenu = 0; //variable de position dans menu principal

int posSousMenu[2] = {0, 0}; // tableau pour stocker les positions de chaque sous-menu

int modeLed = 0;//variable du mode de la LED 13 (0=OFF, 1=ON, 2=Clignote)

boolean etatLed = 0; //état de la LED 13 (0=LOW, 1=HIGH)

int nCAN = 0; //numéro de CAN actuel

String ligne1[2] = {"LED 13 : ", "Analogique : "}; //tableau de titre du menu principal

String ligne2 = "OFF";//chaîne pour la ligne 2 (dépend du menu 1)

unsigned long tpsDep = millis(); //temps départ pour clignotement

void setup() {

//configuration des pins

pinMode(bout1, INPUT);

pinMode(bout2, INPUT);

pinMode(13, OUTPUT);

//initialisation de l'écran LCD

monEcran.begin(16, 2); //on initialise la communication avec 16 colonnes et 2 lignes

monEcran.clear(); // on efface l'écran

}

void loop() {

navigation(); //appel de la fonction pour naviguer dans les menus

affichage(); //appel de la fonction pour rafraîchir l'affichage LCD

allume(); //appel de la fonction pour gérer la LED 13

}

//fonction de navigation dans le menu

void navigation() {

//création de variables d'état pour les boutons

boolean etatBout1 = digitalRead(bout1);

boolean etatBout2 = digitalRead(bout2);

//Boucle pour naviguer dans les menus

if (etatBout1 || etatBout2) { //si l'un des deux boutons est appuyé

if (etatBout1) { // si bouton 1

posMenu = (posMenu + 1) % 2; //on change le menu principal (2 positions)

}

if (etatBout2) { // si bouton 2

switch (posMenu) { //l'action dépend du menu 1

case 0: //si menu LED

modeLed = (modeLed + 1) % 3; //on change le mode de la LED

break;

case 1 : //si menu CAN

nCAN = (nCAN + 1) % 5; //on change le CAN lu

break;

}

}

delay(200); //attente pour éviter les répétitions

}

}

//fonction de rafraîchissement de l'écran LCD

void affichage() {

monEcran.setCursor(0, 0); // on positionne le curseur en haut à gauche

monEcran.print(ligne1[posMenu]); // on écrit le menu de la ligne 1

monEcran.setCursor(0, 1); // on passe à la ligne suivante

//définition du menu pour la LED 13

String mLED[3] = {

"OFF ",

"ON ",

"Clignote "

};

switch (posMenu) { // en fonction du menu 1

case 0: // si menu LED

ligne2 = mLED[modeLed]; //titre pris dans tableau mLED

break;

case 1: // si menu CAN

readCAN(); // appel de la construction de l'affichage

break;

}

monEcran.print(ligne2); // on affiche la ligne du menu 2

}

// fonction de construction de l'affichage de lecture de CAN

void readCAN () {

int val = analogRead(nCAN); // on lit le CAN concerné

ligne2 = "CAN "; //on initialise la chaîne

ligne2 += String(nCAN); //on ajoute le numéro du CAN actuel

ligne2 += " : "; //pour présentation

ligne2 += String(val); // on ajoute la valeur lue du CAN actuel

ligne2 += " "; // on ajoute des espaces pour effacer l'affichage précédent

//la ligne de menu est construite

}

//fonction qui gère la LED 13

void allume() {

unsigned long tpsAct = millis(); //variable du temps actuel

if (tpsAct - tpsDep > 200) { // si plus de 200ms

switch (modeLed) { // on agit en fonction du mode actuel

case 0: // si OFF

etatLed = 0; // on éteint la LED

break;

case 1: // si ON

etatLed = 1; // on allume la LED

break;

case 2: // si Clignote

etatLed = !etatLed; //on inverse l'état de la LED

break;

}

digitalWrite(13, etatLed); //on positionne la LED dans l'état concerné

tpsDep = tpsAct; //on initialise le temps de départ

}

}

Ce programme est un exemple.

Si vous voulez continuer dans votre lancée de pratique, vous pouvez tout à fait imaginer un menu plus conséquent qui permet de modifier la vitesse de clignotement, qui gère plusieurs LED, qui fait varier la vitesse d'un moteur (grâce au PWM), etc.

J'attire votre attention sur les espaces ajoutés en fin de chaîne de caractères pour les titres des menus :

On peut tout à fait effacer l'écran avec la fonction monEcran.clear() ,  à chaque passage de la boucle. Mais cette méthode écrit en fait deux lignes de caractères vides, ce qui provoque un affichage peu agréable.

Le programme permet de lire l'état des CAN en temps réel, il faut donc l'afficher à chaque tour de  loop().

En affichant une chaîne de caractères avec assez d'espaces pour remplir la ligne, on efface ainsi les caractères précédents. C'est une méthode qui améliore la qualité visuelle. Faites l'essai sans mettre les espaces pour voir. ;)

**En résumé :**

Vous avez appris comment connecter un écran LCD à votre carte Arduino et comment l’utiliser comme interface grâce à la bibliothèque LiquidCrystal.  
Vous avez appris à ajouter un menu et des boutons de navigation sur cette interface, et à les gérer avec un programme Arduino.

Finalement, si votre Arduino est alimenté en autonomie (donc non connecté à votre ordinateur), vous avez de quoi le paramétrer et le lire, juste avec deux boutons. Une interface homme/machine assez efficace.

On peut imaginer le paramétrage d'un scénario pour un robot grâce à un menu (suivi de ligne, rebond sur ligne, évitement d'obstacles...) ou la lecture d'informations liées à une station météo maison (température, humidité, lancement ou non d'une pompe d'arrosage...). Ces écrans LCD sont donc des plus utiles en mode autonome !

Le seul inconvénient est le nombre de pins utilisés pour communiquer dans ce mode de connexion. Mais comme je vous l'ai signalé au début, il existe des moyens simples de passer en mode I2C, qui n'utilisent alors que les pins A4 et A5 de l'Arduino.

Je m'arrête là pour les LCD, voyons maintenant les possibilités de commander votre Arduino à distance grâce à un capteur infrarouge et une télécommande simple !