Arduino :: Afficheurs

<http://composants.e44.com/afficheurs/afficheurs-7-segments/#topliste>

<http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/afficheurs-7-segments>

<http://eskimon.fr/98-arduino-205-afficheurs-7-segments>

<http://forum.snootlab.com/viewtopic.php?f=38&t=914>

<http://www.pobot.org/LCD-sur-I2C-simplement.html>

<http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD>

<http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte>

<http://composants.e44.com/afficheurs/afficheurs-lcd/afficheur-lcd-2x16-caracteres-84x44x11mm-AFFLCD2X16.html#Danslamemecategorie>

<http://www.pobot.org/Piloter-des-afficheurs-7-segments.html>

<http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html>

<http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/>

<http://diy.enna.fr/2013/07/15/raspberrypi-i2c/>

<http://arduino103.blogspot.fr/2013/03/des-matrices-led-8x8-adafruit-i2c.html>

<http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK>

<http://www.sonelec-musique.com/logiciels_pic_bases_mp_i2c_pcf8574.html>

<http://home.roboticlab.eu/fr/examples/display/segment_display>

<http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html>

<http://skyduino.wordpress.com/2012/05/01/arduino-de-electronique-avec-capteur-tilt-et-afficheur-7-segments/>

[Arduino :: Afficheurs 1](#_Toc404255431)

[http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/afficheurs-7-segments 5](#_Toc404255432)

[**Première approche : côté électronique** 5](#_Toc404255433)

[**Afficher son premier chiffre !** 9](#_Toc404255434)

[**Techniques d'affichage** 11](#_Toc404255435)

[**Utilisation du décodeur BCD** 15](#_Toc404255436)

[**Utiliser plusieurs afficheurs** 20](#_Toc404255437)

[**Contraintes des évènements** 28](#_Toc404255438)

[http://eskimon.fr/98-arduino-205-afficheurs-7-segments 31](#_Toc404255439)

[Première approche : côté électronique 31](#_Toc404255440)

[Un peu (beaucoup) d’électronique 31](#_Toc404255441)

[Branchement “complet” de l’afficheur 32](#_Toc404255442)

[Afficher son premier chiffre ! 35](#_Toc404255443)

[Schéma de connexion 35](#_Toc404255444)

[Le programme 36](#_Toc404255445)

[Techniques d’affichage 37](#_Toc404255446)

[Les décodeurs “4 bits -> 7 segments” 38](#_Toc404255447)

[L’affichage par alternance 41](#_Toc404255448)

[Utilisation du décodeur BCD 41](#_Toc404255449)

[Utiliser plusieurs afficheurs 46](#_Toc404255450)

[Un peu d’électronique… 47](#_Toc404255451)

[…et de programmation 52](#_Toc404255452)

[Contraintes des évènements 55](#_Toc404255453)

[http://forum.snootlab.com/viewtopic.php?f=38&t=914 59](#_Toc404255454)

[Cristaux Liquide / LCD 62](#_Toc404255455)

[http://www.pobot.org/LCD-sur-I2C-simplement.html 63](#_Toc404255456)

[Le matériel 63](#_Toc404255457)

[Le logiciel 64](#_Toc404255458)

[http://www.mon-club-elec.fr/pmwiki\_mon\_club\_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD 66](#_Toc404255459)

[Préparation d'un afficheur LCD pour une utilisation simplifiée avec une carte Arduino, PIC, etc.. 66](#_Toc404255460)

[http://www.mon-club-elec.fr/pmwiki\_mon\_club\_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte.. 70](#_Toc404255461)

[1. Présentation 71](#_Toc404255462)

[2. Matériel Nécessaire 71](#_Toc404255463)

[2.1 L'espace de développement Arduino 71](#_Toc404255464)

[2.2 Le matériel suivant pour réaliser le montage associé 71](#_Toc404255465)

[3. Instructions de montage 71](#_Toc404255466)

[4. Le schéma théorique du montage 72](#_Toc404255467)

[5. Le circuit du montage 73](#_Toc404255468)

[6. Au niveau de la partie déclarative : 73](#_Toc404255469)

[7. Au niveau de la fonction d'initialisation setup( ) : 74](#_Toc404255470)

[8. Au niveau de la boucle principale, la fonction loop ( ) : 75](#_Toc404255471)

[9. Mise en oeuvre du programme 75](#_Toc404255472)

[9.1 Préparation du montage et programmation de la carte Arduino : 75](#_Toc404255473)

[9.2 Fonctionnement 76](#_Toc404255474)

[10. Le programme complet en langage Arduino : 76](#_Toc404255475)

[Annexes 78](#_Toc404255476)

[http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/introduire-le-temps 78](#_Toc404255477)

[Comment faire ? 78](#_Toc404255478)

[Faire clignoter un groupe de LED 82](#_Toc404255479)

[Réaliser un chenillard 86](#_Toc404255480)

[Fonction millis() 88](#_Toc404255481)

[http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/ajouter-des-sorties-numeriques-a-l-arduino 90](#_Toc404255482)

[Ajouter des sorties (numériques) à l'Arduino 90](#_Toc404255483)

[Présentation du 74HC595 91](#_Toc404255484)

[Programmons pour utiliser ce composant 95](#_Toc404255485)

[Exercices : encore des chenillards ! 103](#_Toc404255486)

[Pas assez ? Augmenter encore ! 105](#_Toc404255487)

[http://www.pobot.org/Piloter-des-afficheurs-7-segments.html 110](#_Toc404255488)

[Piloter des afficheurs 7 segments 110](#_Toc404255489)

[Exemples et cahier des charges 111](#_Toc404255490)

[Quelques solutions possibles 111](#_Toc404255491)

[Réalisation du chronomètre 112](#_Toc404255492)

[Code source 112](#_Toc404255493)

[http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html 115](#_Toc404255494)

[Ecran CLCDxxx sur I2C avec Arduino 115](#_Toc404255495)

[L’écran 116](#_Toc404255496)

[Le code 116](#_Toc404255497)

[Le résultat 116](#_Toc404255498)

[http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/ 117](#_Toc404255499)

[RaspberryPI: Afficheur 7 segments avec I2C 117](#_Toc404255500)

[Activation de I2C 118](#_Toc404255501)

[Installation des paquets 119](#_Toc404255502)

[Activation des modules linux 119](#_Toc404255503)

[Vérification 119](#_Toc404255504)

[Animation de test 119](#_Toc404255505)

[Récupération des sources 119](#_Toc404255506)

[Test d’affichage sur la matrice 119](#_Toc404255507)

[Pour aller plus loin 120](#_Toc404255508)

[Afficher 12:34 120](#_Toc404255509)

[Afficher °C 121](#_Toc404255510)

[http://diy.enna.fr/2013/07/15/raspberrypi-i2c/ 122](#_Toc404255511)

[RaspberryPI: Matrice de LED avec I2C 122](#_Toc404255512)

[Activation de I2C 123](#_Toc404255513)

[Installation des paquets 124](#_Toc404255514)

[Activation des modules linux 124](#_Toc404255515)

[Vérification 124](#_Toc404255516)

[Animation de test 124](#_Toc404255517)

[Récupération des sources 124](#_Toc404255518)

[Test d’affichage sur la matrice 124](#_Toc404255519)

[Afficher un nuage 125](#_Toc404255520)

[Pour aller plus loin 126](#_Toc404255521)

[http://arduino103.blogspot.fr/2013/03/des-matrices-led-8x8-adafruit-i2c.html 126](#_Toc404255522)

[Des matrices LED 8x8 AdaFruit (I2C) disponibles chez MCHobby 126](#_Toc404255523)

[http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK 129](#_Toc404255524)

[http://www.sonelec-musique.com/logiciels\_pic\_bases\_mp\_i2c\_pcf8574.html 132](#_Toc404255525)

[http://home.roboticlab.eu/fr/examples/display/segment\_display 141](#_Toc404255526)

[Afficheur 7 segments à LED 141](#_Toc404255527)

[Théorie 141](#_Toc404255528)

[Pratique 141](#_Toc404255529)

[http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit\_10TS3.html 145](#_Toc404255530)

[http://skyduino.wordpress.com/2012/05/01/arduino-de-electronique-avec-capteur-tilt-et-afficheur-7-segments/ 155](#_Toc404255531)

[Dé électronique avec capteur tilt et afficheur 7 segments 155](#_Toc404255532)

# http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/afficheurs-7-segments

Vous connaissez les afficheurs 7 segments ? Ou alors vous ne savez pas que ça s'appelle comme ça ? Il s'agit des petites lumières qui forment le chiffre 8 et qui sont de couleur rouge ou verte, la plupart du temps, mais peuvent aussi être bleus, blancs, etc. On en trouve beaucoup dans les radio-réveils, car ils servent principalement à afficher l'heure. Autre particularité, non seulement de pouvoir afficher des chiffres (0 à 9), ils peuvent également afficher certaines lettre de l'alphabet.

**Matériel**  
Pour ce chapitre, vous aurez besoin de :

* Un (et plus) afficheur 7 segments (évidemment)
* 8 résistances de 330\Omega
* Un (ou deux) décodeurs BCD 7 segments
* Une carte Arduino ! Mais dans un premier temps on va d'abord bien saisir le truc avant de faire du code :)

Nous allons commencer par une découverte de l'afficheur, comment il fonctionne et comment le branche-t-on. Ensuite nous verrons comment l'utiliser avec la carte Arduino. Enfin, le chapitre suivant amènera un TP résumant les différentes parties vues.

**Première approche : côté électronique**

**Un peu (beaucoup) d'électronique**

Comme son nom l'indique, l'afficheur 7 segments possède... 7 segments. Mais un segment c'est quoi au juste ? Et bien c'est une portion de l'afficheur, qui est allumée ou éteinte pour réaliser l'affichage. Cette portion n'est en fait rien d'autre qu'une LED qui au lieu d'être ronde comme d'habitude est plate et encastré dans un boiter. On dénombre donc 8 portions en comptant le point de l'afficheur (mais il ne compte pas en tant que segment à part entière car il n'est pas toujours présent). Regardez à quoi ça ressemble :

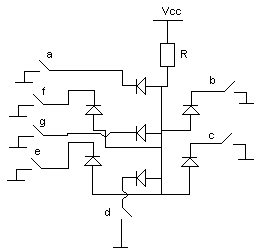
Image utilisateur

*Afficheur 7 segments*

**Des LED, encore des LED**

Et des LED, il y en a ! Entre 7 et 8 selon les modèles (c'est ce que je viens d'expliquer), voir beaucoup plus, mais on ne s'y attardera pas dessus.

Voici un schéma vous présentant un modèle d'afficheur sans le point (qui au final est juste une LED supplémentaire rappelez-vous) :



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

Comme vous le voyez sur ce schéma, toutes les LED possèdent une broche commune, reliée entre elle. Selon que cette broche est la cathode ou l'anode on parlera d'afficheur à cathode commune ou... anode commune (vous suivez ?). Dans l'absolu, ils fonctionnent de la même façon, seule la manière de les brancher diffère (actif sur état bas ou sur état haut).

**Cathode commune ou Anode commune**

Dans le cas d'un afficheur à cathode commune, toutes les cathodes sont reliées entre elles en un seul point lui-même connecté à la masse. Ensuite, chaque anode de chaque segment sera reliée à une broche de signal. Pour allumer chaque segment, le signal devra être une tension positive. En effet, si le signal est à 0, il n'y a pas de différence de potentiel entre les deux broches de la LED et donc elle ne s'allumera pas !

Si nous sommes dans le cas d'une anode commune, les anodes de toutes les LED sont reliées entre elles en un seul point qui sera connecté à l'alimentation. Les cathodes elles seront reliées une par une aux broches de signal. En mettant une broche de signal à 0, le courant passera et le segment en question s'allumera. Si la broche de signal est à l'état haut, le potentiel est le même de chaque côté de la LED, donc elle est bloquée et ne s'allume pas !

Que l'afficheur soit à anode ou à cathode commune, on doit toujours prendre en compte qu'il faut ajouter une résistance de limitation de courant entre la broche isolée et la broche de signal. Traditionnellement, on prendra une résistance de 330 ohms pour une tension de +5V, mais cela se calcul (cf. chapitre 1, partie 2). Si vous voulez augmenter la luminosité, il suffit de diminuer cette valeur. Si au contraire vous voulez diminuer la luminosité, augmenter la résistance.

**Choix de l'afficheur**

Pour la rédaction j'ai fait le choix d'utiliser des afficheurs à anode commune et ce n'est pas anodin. En effet et on l'a vu jusqu'à maintenant, on branche les LED du +5V vers la broche de la carte Arduino. Ainsi, dans le cas d'un afficheur à anode commune, les LED seront branchés d'un côté au +5V, et de l'autre côté aux broches de signaux. Ainsi, pour allumer un segment on mettra la broche de signal à 0 et on l'éteindra en mettant le signal à 1. On a toujours fait comme ça depuis le début, ça ne vous posera donc aucun problème. ;)

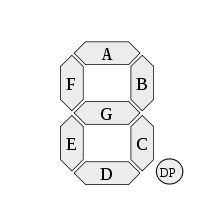
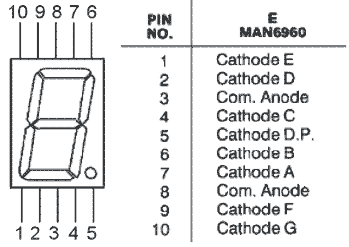
**Branchement "complet" de l'afficheur**

Nous allons maintenant voir comment brancher l'afficheur à anode commune.

**Présentation du boîtier**

Les afficheurs 7 segments se présentent sur un *boîtier* de type DIP 10. Le format DIP régie l'espacement entre les différentes broches du circuit intégré ainsi que d'autres contraintes (présence d'échangeur thermique etc...). Le chiffre 10 signifie qu'il possède 10 broches (5 de part et d'autre du boitier).

Voici une représentation de ce dernier (à gauche) :



Voici la signification des différentes broches :

1. LED de la cathode E
2. LED de la cathode D
3. Anode commune des LED
4. LED de la cathode C
5. (facultatif) le point décimal.
6. LED de la cathode B
7. LED de la cathode A
8. Anode commune des LED
9. LED de la cathode F
10. LED de la cathode G

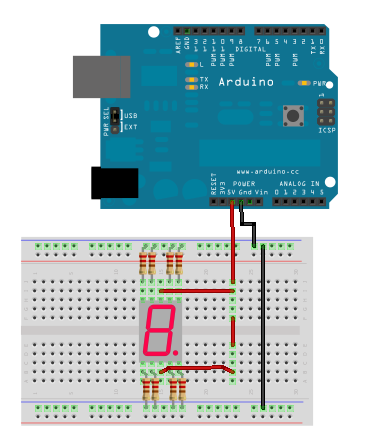
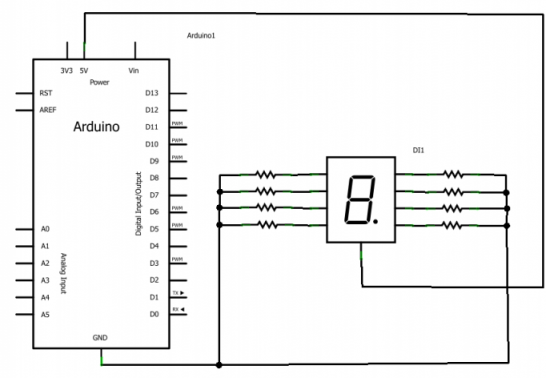
Pour allumer un segment c'est très simple, il suffit de le relier à la masse !

Nous cherchons à allumer les LED de l'afficheur, il est donc impératif de ne pas oubliez les résistances de limitations de courant !

**Exemple**

Pour commencer, vous allez tout d'abord mettre l'afficheur à cheval sur la plaque d'essai (breadboard). Ensuite, trouvez la broche représentant l'anode commune et reliez la à la future colonne du +5V. Prochaine étape, mettre une résistance de 330 \Omega sur chaque broche de signal. Enfin, reliez quelques une de ces résistances à la masse. Si tous se passe bien, les segments reliés à la masse via leur résistance doivent s'allumer lorsque vous alimentez le circuit.

Voici un exemple de branchement :



Dans cet exemple de montage, vous verrez que tous les segment de l'afficheur s'allument ! Vous pouvez modifier le montage en déconnectant quelques unes des résistance de la masse et afficher de nombreux caractères.

Pensez à couper l'alimentation lorsque vous changer des fils de place. Les composants n'aiment pas forcément être (dé)branchés lorsqu'ils sont alimentés. Vous pourriez éventuellement leur causer des dommages.

**Seulement 7 segments mais plein de caractère(s) !**

Vous l'avez peut-être remarqué avec "l'exercice" précédent, un afficheurs 7 segments ne se limite pas à afficher juste des chiffres. Voici un tableau illustrant les caractères possibles et quels segments allumés. Attention, il est possible qu'il manque certains caractères !

| **Caractère** | **seg. A** | **seg. B** | **seg. C** | **seg. D** | **seg. E** | **seg. F** | **seg. G** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | x | x | x | x | x | x |  |
| 1 |  | x | x |  |  |  |  |
| 2 | x | x |  | x | x |  | x |
| 3 | x | x | x | x |  |  | x |
| 4 |  | x | x |  |  | x | x |
| 5 | x |  | x | x |  | x | x |
| 6 | x |  | x | x | x | x | x |
| 7 | x | x | x |  |  |  |  |
| 8 | x | x | x | x | x | x | x |
| 9 | x | x | x | x |  | x | x |
| A | x | x | x |  | x | x | x |
| b |  |  | x | x | x | x | x |
| C | x |  |  | x | x | x |  |
| d |  | x | x | x | x | x |  |
| E | x |  |  | x | x | x | x |
| F | x |  |  |  | x | x | x |
| H |  | x | x |  | x | x | x |
| I |  | x | x |  |  |  |  |
| J |  | x | x | x | x |  |  |
| L |  |  |  | x | x | x |  |
| o |  |  | x | x | x |  | x |
| P | x | x |  |  | x | x | x |
| S | x |  | x | x |  | x | x |
| t |  |  |  |  | x | x | x |
| U |  | x | x | x | x | x |  |
| y |  | x | x | x |  | x | x |
| ° | x | x |  |  |  | x | x |

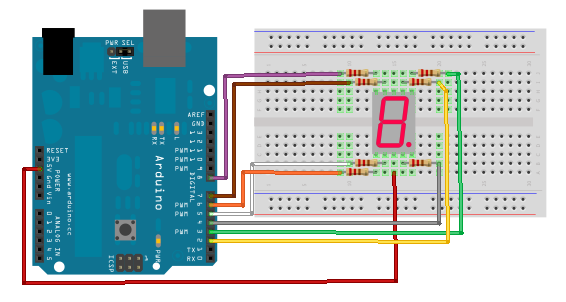
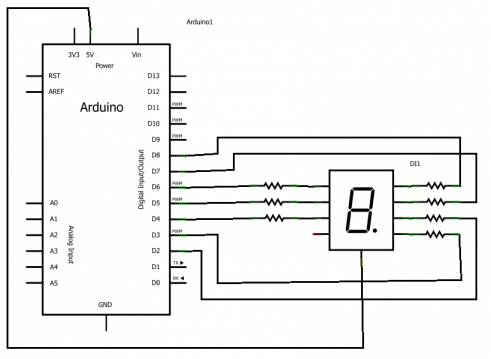
Aidez vous de ce tableau lorsque vous aurez à coder l'affichage de caractères ! ;)

**Afficher son premier chiffre !**

Pour commencer, nous allons prendre en main un afficheur et lui faire s'afficher notre premier chiffre ! C'est assez simple et ne requiert qu'un programme très simple, mais un peu rébarbatif.

**Schéma de connexion**

Je vais reprendre le schéma précédent, mais je vais connecter chaque broche de l'afficheur à une sortie de la carte Arduino. Comme ceci :



Vous voyez donc que chaque LED de l'afficheur va être commandée séparément les unes des autres. Il n'y a rien de plus à faire, si ce n'est qu'à programmer...

**Le programme**

L'objectif du programme va être d'afficher un chiffre. Eh bien... c'est partit !

Quoi ?! Vous voulez de l'aide ? o_OBen je vous ai déjà tout dit y'a plus qu'à faire. En plus vous avez un tableau avec lequel vous pouvez vous aider pour afficher votre chiffre.

Cherchez, je vous donnerais la solution ensuite.

Solution :

/\* On assigne chaque LED à une broche de l'arduino \*/

const int A = 2;

const int B = 3;

const int C = 4;

const int D = 5;

const int E = 6;

const int F = 7;

const int G = 8;

//notez que l'on ne gère pas l'affichage du point, mais vous pouvez le rajouter si cela vous chante ^^

void setup()

{

   //définition des broches en sortie

   pinMode(A, OUTPUT);

   pinMode(B, OUTPUT);

   pinMode(C, OUTPUT);

   pinMode(D, OUTPUT);

   pinMode(E, OUTPUT);

   pinMode(F, OUTPUT);

   pinMode(G, OUTPUT);

   //mise à l'état HAUT de ces sorties pour éteindre les LED de l'afficheur

   digitalWrite(A, HIGH);

   digitalWrite(B, HIGH);

   digitalWrite(C, HIGH);

   digitalWrite(D, HIGH);

   digitalWrite(E, HIGH);

   digitalWrite(F, HIGH);

   digitalWrite(G, HIGH);

}

void loop()

{

   //affichage du chiffre 5, d'après le tableau précédent

   digitalWrite(A, LOW);

   digitalWrite(B, HIGH);

   digitalWrite(C, LOW);

   digitalWrite(D, LOW);

   digitalWrite(E, HIGH);

   digitalWrite(F, LOW);

   digitalWrite(G, LOW);

}

Vous le voyez par vous-même, c'est un code hyper simple. Essayez de le bidouiller pour afficher des messages, par exemple, en utilisant les fonctions introduisant le temps. Ou bien compléter ce code pour afficher tous les chiffres, en fonction d'une variable définie au départ (ex: var = 1, affiche le chiffre 1 ; etc.).

**Techniques d'affichage**

Vous vous en doutez peut-être, lorsque l'on veut utiliser plusieurs afficheur il va nous falloir beaucoup de broches. Imaginons, nous voulons afficher un nombre entre 0 et 99, il nous faudra utiliser deux afficheurs avec 2\*7 = 14 broches connectées sur la carte Arduino. Rappel : une carte Arduino UNO possède... 14 broches entrées/sorties classiques. Si on ne fais rien d'autre que d'utiliser les afficheurs, cela ne nous gène pas, cependant, il est fort probable que vous serez amener à utiliser d'autres entrées avec votre carte Arduino. Mais si on ne libère pas de place vous serez embêté. Nous allons donc voir deux techniques qui, une fois cumulées, vont nous permettre d'utiliser seulement 4 broches pour obtenir le même résultat qu'avec 14 broches !

**Les décodeurs "4 bits -> 7 segments"**

La première technique que nous allons utiliser met en œuvre un circuit intégré. Vous vous souvenez quand je vous ai parlé de ces bêtes là ? Oui, c'est le même type que le microcontrôleur de la carte Arduino. Cependant, le circuit que nous allons utiliser ne fait pas autant de choses que celui sur votre carte Arduino.

**Décodeur BCD -> 7 segments**

C'est le nom du circuit que nous allons utiliser. Son rôle est simple. Vous vous souvenez des conversions ? Pour passer du binaire au décimal ? Et bien c'est le moment de vous en servir, donc si vous ne vous rappelez plus de ça, allez revoir un peu [le cours](http://sciences.siteduzero.com/tutoriel-3-457255-1-quelques-bases-elementaires.html#ss_part_4).

Je disais donc que son rôle est simple. Et vous le constaterez par vous même, il va s'agir de convertir du binaire codé sur 4 bits vers un "code" utilisé pour afficher les chiffres. Ce code correspond en quelque sorte au tableau précédemment évoqué.

**Principe du décodeur**

Sur un afficheur 7 segments, on peut représenter aisément les chiffres de 0 à 9 (et en insistant un peu les lettres de A à F). En informatique, pour représenter ces chiffres, il nous faut au maximum 4 bits. Comme vous êtes des experts et que vous avez bien lu la partie sur le binaire, vous n'avez pas de mal à le comprendre. (0000)2 fera (0)10 et (1111)2 fera (15)10 ou (F)16. Pour faire 9 par exemple on utilisera les bits 1001.

En partant de se constat, des ingénieurs ont inventé un composant au doux nom de "décodeur" ou "driver" 7 segments. Il reçoit sur 4 broches les 4 bits de la valeur à afficher, et sur 7 autres broches ils pilotent les segments pour afficher ladite valeur. Ajouter à cela une broche d'alimentation et une broche de masse on obtient 13 broches ! Et ce n'est pas fini. La plupart des circuits intégrés de type décodeur possède aussi une broche d'activation et une broche pour tester si tous les segments fonctionnent.

**Choix du décodeur**

Nous allons utiliser le composant nommé MC14543B comme exemple. Tout d'abord, ouvrez ce lien dans un nouvel onglet, il vous menera directement vers le pdf du décodeur :

[Datasheet du MC14543B](http://www.datasheetcatalog.org/datasheet2/4/09lwz6g28frlr15ayl6w0srxwz7y.pdf)

Les datasheets se composent souvent de la même manière. On trouve tout d'abord un résumé des fonctions du produit puis un schéma de son boîtier. Dans notre cas, on voit qu'il est monté sur un DIP 16 (DIP : Dual Inline Package, en gros "boîtier avec deux lignes de broches"). Si l'on continue, on voit la **table de vérité** faisant le lien entre les signaux d'entrées (INPUT) et les sorties (OUTPUT). On voit ainsi plusieurs choses :

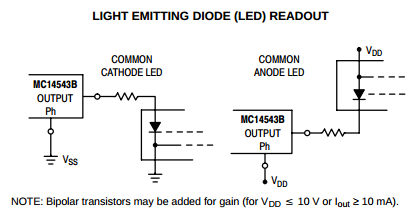
* Si l'on met la broche Bl (Blank, n°7) à un, toutes les sorties passent à zéro. En effet, comme son nom l'indique cette broche sert à effacer l'état de l'afficheur. Si vous ne voulez pas l'utiliser il faut donc la connecter à la masse pour la désactiver.
* Les entrées A, B, C et D (broches 5,3,2 et 4 respectivement) sont actives à l'état HAUT. Les sorties elles sont actives à l'état BAS (pour piloter un afficheur à anode commune) **OU** HAUT selon l'état de la broche PH (6). C'est là un gros avantage de ce composant, il peut inverser la logique de la sortie, le rendant alors compatible avec des afficheurs à anode commune (broche PH à l'état 1) ou cathode commune (Ph = 0)
* La broche BI/RBO (n°4) sers à inhiber les entrées. On ne s'en servira pas et donc on la mettra à l'état HAUT (+5V)
* LD (n°1) sert à faire une mémoire de l'état des sorties, on ne s'en servira pas ici
* Enfin, les deux broches d'alimentation sont la 8 (GND/VSS, masse) et la 16 (VCC, +5V)

N'oubliez pas de mettre des résistances de limitations de courant entre chaque segment et la broche de signal du circuit!

**Fonctionnement**

C'est bien beau tout ça mais comment je lui dis au décodeur d'afficher le chiffre 5 par exemple ?

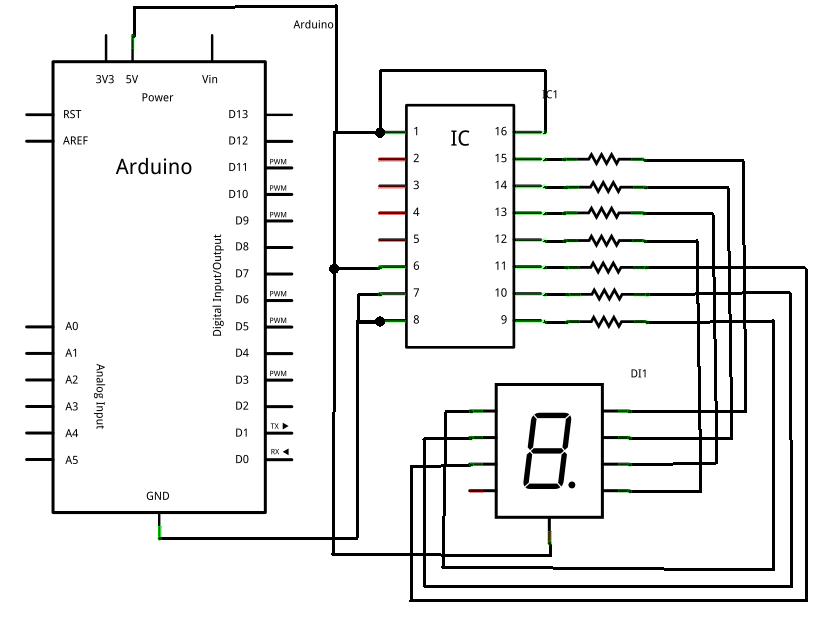
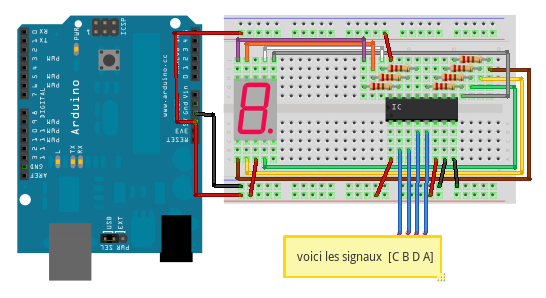
Il suffit de regarder le datasheet et sa table de vérité (c'est le tableau avec les entrées et les sorties). Ce que reçoit le décodeur sur ses entrées (A, B, C et D) défini les états de ses broches de sortie (a,b,c,d,e,f et g). C'est tout ! Donc, on va donner un code binaire sur 4 bits à notre décodeur et en fonction de ce code, le décodeur affichera le caractère voulu. En plus le fabricant est sympa, il met à disposition des notes d'applications à la page 6 pour bien brancher le composant :

Branchement du MC14543B

On voit alors qu'il suffit simplement de brancher la résistance entre le CI et les segments et s'assurer que PH à la bonne valeur et c'est tout !

En titre d'exercice afin de vous permettre de mieux comprendre, je vous propose de changer les états des entrées A, B, C et D du décodeur pour observer ce qu'il affiche.

Après avoir réaliser votre schéma, regarder s'il correspond avec celui présent dans cette balise secrète. Cela vous évitera peut-être un mauvais branchement, qui sait ?

Montage 7 segments, schémaMontage 7 segments, breadboard

**L'affichage par alternance**

La seconde technique est utilisée dans le cas où l'on veut faire un affichage avec plusieurs afficheurs. Elle utilise le phénomène de [persistance rétinienne](http://fr.wikipedia.org/wiki/Persistance_r%C3%A9tinienne). Pour faire simple, c'est grâce à cela que le cinéma vous parait fluide. On change une image toutes les 40 ms et votre œil n'a pas le temps de le voir, donc les images semble s'enchainer sans transition. Bref...

Ici, la même stratégie sera utilisée. On va allumer un afficheur un certain temps, puis nous allumerons l'autre en éteignant le premier. Cette action est assez simple à réaliser, mais nécessite l'emploi de deux broche supplémentaires, de quatre autres composants et d'un peu de code. Nous l'étudierons un petit peu plus tard, lorsque nous saurons géré un afficheur seul.

**Utilisation du décodeur BCD**

Nous y sommes, nous allons (enfin) utiliser la carte Arduino pour faire un affichage plus poussé qu'un unique afficheur. Pour cela, nous allons très simplement utiliser le montage précédent composé du décodeur BCD, de l'afficheur 7 segments et bien entendu des résistances de limitations de courant pour les LED de l'afficheur. Je vais vous montrer deux techniques qui peuvent être employées pour faire le programme.

**Initialisation**

Vous avez l'habitude maintenant, nous allons commencer par définir les différentes broches d'entrées/sorties. Pour débuter (et conformément au schéma), nous utiliserons seulement 4 broches, en sorties, correspondantes aux entrées du décodeur 7 segments.

Voici le code pouvant traduire cette explication :

const int bit\_A = 2;

const int bit\_B = 3;

const int bit\_C = 4;

const int bit\_D = 5;

void setup()

{

   //on met les broches en sorties

   pinMode(bit\_A, OUTPUT);

   pinMode(bit\_B, OUTPUT);

   pinMode(bit\_C, OUTPUT);

   pinMode(bit\_D, OUTPUT);

   //on commence par écrire le chiffre 0, donc toutes les sorites à l'état bas

   digitalWrite(bit\_A, LOW);

   digitalWrite(bit\_B, LOW);

   digitalWrite(bit\_C, LOW);

   digitalWrite(bit\_D, LOW);

}

Ce code permet juste de déclarer les quatre broches à utiliser, puis les affectes en sorties. On les met ensuite toutes les quatre à zéro. Maintenant que l'afficheur est prêt, nous allons pouvoir commencer à afficher un chiffre !

**Programme principal**

Si tout se passe bien, en ayant la boucle vide pour l'instant vous devriez voir un superbe 0 sur votre afficheur. Nous allons maintenant mettre en place un petit programme pour afficher les nombres de 0 à 9 en les incrémentant (à partir de 0) toutes les secondes. C'est donc un compteur.

Pour cela, on va utiliser une boucle, qui comptera de 0 à 9. Dans cette boucle, on exécutera appellera la fonction affichage() qui s'occupera donc de l'affichage (belle démonstration de ce qui est une évidence :p^^).

void loop()

{

    char i=0; //variable "compteur"

    for(i=0; i<10; i++)

    {

        affichage(i); //on appel la fonction d'affichage

        delay(1000); //on attend 1 seconde

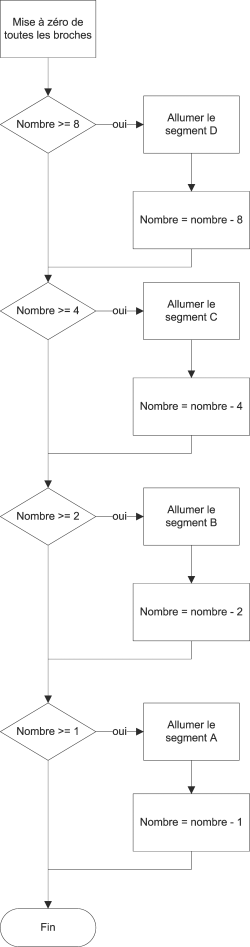
    }

}

**Fonction d'affichage**

Nous touchons maintenant au but ! Il ne nous reste plus qu'à réaliser la fonction d'affichage pour pouvoir convertir notre variable en chiffre sur l'afficheur. Pour cela, il existe différentes solutions. Nous allons en voir ici une qui est assez simple à mettre en œuvre mais qui nécessite de bien être comprise.

Dans cette méthode, on va faire des opérations mathématiques (tout de suite c'est moins drôle :lol:) successives pour déterminer quels bits mettre à l'état haut. Rappelez-vous, nous avons quatre broches à notre disposition, avec chacune un poids différent (8, 4, 2 et 1). En combinant ces différentes broches ont peu obtenir n'importe quel nombre de 0 à 15. Voici une démarche mathématique envisageable :

Organigramme décodeur 7 segments

On peut coder cette méthode de manière assez simple et direct, en suivant cet organigramme :

//fonction écrivant sur un seul afficheur

void afficher(char chiffre)

{

   //on met à zéro tout les segments

   digitalWrite(bit\_A, LOW);

   digitalWrite(bit\_B, LOW);

   digitalWrite(bit\_C, LOW);

   digitalWrite(bit\_D, LOW);

   //On allume les bits nécessaires

   if(chiffre >= 8)

   {

      digitalWrite(bit\_D, HIGH);

      chiffre = chiffre - 8;

   }

   if(chiffre >= 4)

   {

      digitalWrite(bit\_C, HIGH);

      chiffre = chiffre - 4;

   }

   if(chiffre >= 2)

   {

      digitalWrite(bit\_B, HIGH);

      chiffre = chiffre - 2;

   }

   if(chiffre >= 1)

   {

      digitalWrite(bit\_A, HIGH);

      chiffre = chiffre - 1;

   }

}

Quelques explications s'imposent...

Le code gérant l'affichage réside sur les valeurs binaires des chiffres. Rappelons les valeurs binaires des chiffres :

| **Chiffre** | **DCBA** |
| --- | --- |
| 0 | (0000)2 |
| 1 | (0001)2 |
| 2 | (0010)2 |
| 3 | (0011)2 |
| 4 | (0100)2 |
| 5 | (0101)2 |
| 6 | (0110)2 |
| 7 | (0111)2 |
| 8 | (1000)2 |
| 9 | (1001)2 |

D'après ce tableau, si on veut le chiffre 8, on doit allumer le segment D, car 8 s'écrit (1000)2 ayant pour segment respectif DCBA. Soit D=1, C=0, B=0 et A=0.

En suivant cette logique, on arrive à déterminer les entrées du décodeur qui sont à mettre à l'état HAUT ou BAS.

D'une manière plus lourde, on aurait pu écrire un code ressemblant à ça :

//fonction écrivant sur un seul afficheur

void afficher(char chiffre)

{

   switch(chiffre)

   {

      case 0 :

         digitalWrite(bit\_A, LOW);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, LOW);

         break;

      case 1 :

         digitalWrite(bit\_A, HIGH);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, LOW);

         break;

      case 2 :

         digitalWrite(bit\_A, LOW);

         digitalWrite(bit\_B, HIGH);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, LOW);

         break;

      case 3 :

         digitalWrite(bit\_A, HIGH);

         digitalWrite(bit\_B, HIGH);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, LOW);

         break;

      case 4 :

         digitalWrite(bit\_A, LOW);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, HIGH);

         digitalWrite(bit\_D, LOW);

         break;

      case 5 :

         digitalWrite(bit\_A, HIGH);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, HIGH);

         digitalWrite(bit\_D, LOW);

         break;

      case 6 :

         digitalWrite(bit\_A, LOW);

         digitalWrite(bit\_B, HIGH);

         digitalWrite(bit\_C, HIGH);

         digitalWrite(bit\_D, LOW);

         break;

      case 7 :

         digitalWrite(bit\_A, HIGH);

         digitalWrite(bit\_B, HIGH);

         digitalWrite(bit\_C, HIGH);

         digitalWrite(bit\_D, LOW);

         break;

      case 8 :

         digitalWrite(bit\_A, LOW);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, HIGH);

         break;

      case 9 :

         digitalWrite(bit\_A, HIGH);

         digitalWrite(bit\_B, LOW);

         digitalWrite(bit\_C, LOW);

         digitalWrite(bit\_D, HIGH);

         break;

   }

}

Mais, c'est bien trop lourd à écrire. Enfin c'est vous qui voyez. ;)

**Utiliser plusieurs afficheurs**

Maintenant que nous avons affiché un chiffre sur un seul afficheur, nous allons pouvoir apprendre à en utiliser plusieurs (avec un minimum de composants en plus !). Comme expliqué précédemment, la méthode employée ici va reposer sur le principe de la persistance rétinienne, qui donnera *l'impression* que les deux afficheurs fonctionnent en *même temps*.

**Problématique**

Nous souhaiterions utiliser deux afficheurs, mais nous ne disposons que de seulement 6 broches sur notre Arduino, le reste des broches étant utilisé pour une autre application. Pour réduire le nombre de broches, on peut d'ores et déjà utilisé un décodeur BCD, ce qui nous ferait 4 broches par afficheurs, soit 8 broches au total. Bon, ce n'est toujours pas ce que l'on veut. Et si on connectait les deux afficheurs ensemble, en parallèle, sur les sorties du décodeur ? Oui mais dans ce cas, on ne pourrait pas afficher des chiffres différents sur chaque afficheur. Tout à l'heure, je vous ai parlé de *commutation*. Oui, la seule solution qui soit envisageable est d'allumer un afficheur et d'éteindre l'autre tout en les connectant ensemble sur le même décodeur. Ainsi un afficheur s'allume, il affiche le chiffre voulu, puis il s'éteint pour que l'autre puisse s'allumer à son tour. Cette opération est en fait un clignotement de chaque afficheur par alternance.

**Un peu d'électronique...**

Pour faire commuter nos deux afficheurs, vous allez avoir besoin d'un nouveau composant, j'ai nommé : le **transistor** !

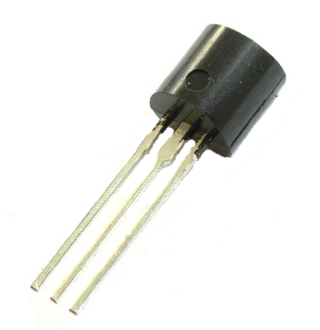
Transistor ? J'ai entendu dire qu'il y en avait plusieurs milliards dans nos ordinateurs ?

Et c'est tout à fait vrai. Des transistors, il en existe de différents types et pour différentes applications : amplification de courant/tension, commutation, etc. répartis dans plusieurs familles. Bon je ne vais pas faire trop de détails, si vous voulez en savoir plus, allez lire la première partie de ce chapitre (*lien à rajouter, en attente de la validation du chapitre en question*).

**Le transistor bipolaire : présentation**

Je le disais, je ne vais pas faire de détails. On va voir comment fonctionne un transistor bipolaire selon les besoins de notre application, à savoir, faire commuter les afficheurs.

Un transistor, cela ressemble à ça :

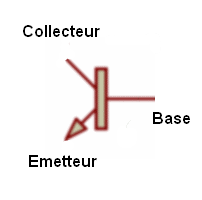


*Photo d'un transistor*

Pour notre application, nous allons utiliser des **transistors bipolaires**. Je vais vous expliquer comment cela fonctionne.

Déjà, vous pouvez observer qu'un transistor possède trois pattes. Cela n'est pas de la moindre importance, au contraire il s'agit là d'une chose essentielle ! En fait, le transistor bipolaire à une broche d'entrée (**collecteur**), une broche de sortie (**émetteur**) et une broche de commande (**base**).

Son symbole est le suivant :



Ce symbole est celui d'un **transistor bipolaire de type NPN**. Il en existe qui sont de **type PNP**, mais ils sont beaucoup moins utilisés que les NPN. Quoi qu'il en soit, nous n'utiliserons que des transistors NPN dans ce chapitre.

**Fonctionnement en commutation du transistor bipolaire**

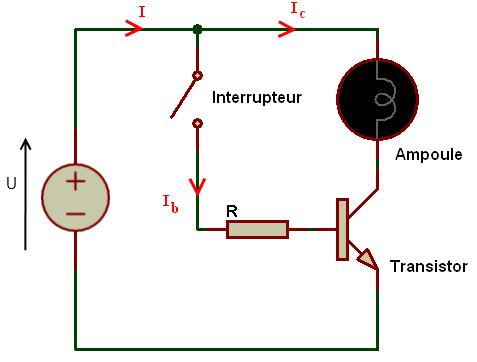
Pour faire simple, le transistor bipolaire NPN (c'est la dernière fois que je précise ce point) est un **interrupteur commandé en courant**.

Ceci est une présentation très vulgarisée et simplifiée sur le transistor pour l'utilisation que nous en ferons ici. Les usages et possibilités des transistors sont très nombreux et ils mériteraient un big-tuto à eux seuls ! Si vous voulez plus d'informations, rendez-vous sur le cours sur l'électronique ou approfondissez en cherchant des tutoriels sur le web. ;)

C'est tout ce qu'il faut savoir, pour ce qui est du fonctionnement. Après, on va voir ensemble comment l'utiliser et sans le faire griller ! ^^

**Utilisation générale**

On peut utiliser notre transistor de deux manières différentes (pour notre application toujours, mais on peut bien évidemment utiliser le transistor avec beaucoup plus de flexibilités). A commencer par le câblage :

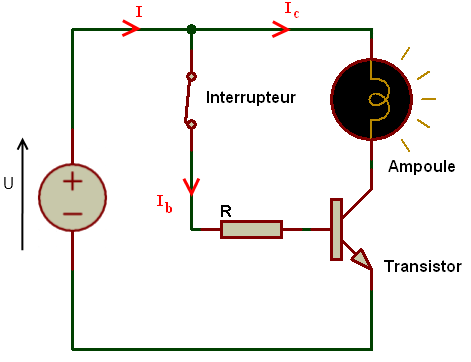


*Câblage du transistor en commutation*

Dans le cas présent, le collecteur (qui est l'entrée du transistor) se trouve être après l'ampoule, elle-même connectée à l'alimentation. L'émetteur (broche où il y a la flèche) est relié à la masse du montage. Cette disposition est "universelle", on ne peut pas inverser le sens de ces broches et mettre le collecteur à la place de l'émetteur et vice versa. Sans quoi, le montage ne fonctionnerait pas.

Pour le moment, l'ampoule est éteinte car le transistor ne conduit pas. On dit qu'il est **bloqué** et empêche donc le courant I\_C de circuler à travers l'ampoule. Soit I\_C = 0 car I\_B = 0.

A présent, appuyons sur l'interrupteur :



*L'ampoule est allumée*

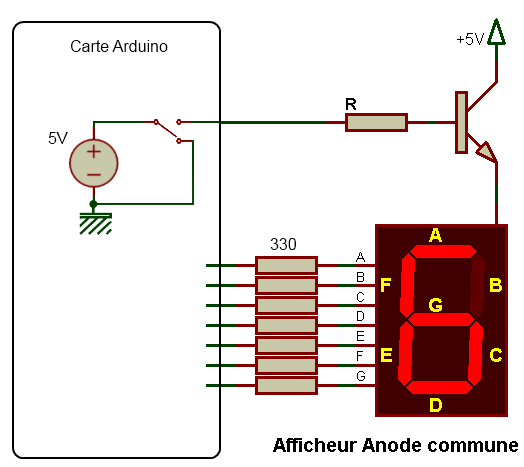
Que se passe-t-il ? Eh bien la base du transistor, qui était jusqu'à présent "en l'air", est parcourue par un courant électrique. Cette cause à pour conséquence de rendre le transistor **passant** ou **saturé** et permet au courant de s'établir à travers l'ampoule. Soit I\_C e 0 car I\_B e 0.

La résistance sur la base du transistor permet de le protéger des courants trop forts. Plus la résistance est de faible valeur, plus l'ampoule sera lumineuse. A l'inverse, une résistance trop forte sur la base du transistor pourra l'empêcher de conduire et de faire s'allumer l'ampoule. Rassurez\_vous, je vous donnerais les valeurs de résistances à utiliser. ;)

**Utilisation avec nos afficheurs**

Voyons un peu comment on va pouvoir utiliser ce transistor avec notre Arduino.

La carte Arduino est en fait le générateur de tension (schéma précédent) du montage. Elle va définir si sa sortie est de 0V (transistor bloqué) ou de 5V (transistor saturé). Ainsi, on va pouvoir allumer ou éteindre les afficheurs. Voilà le modèle équivalent de la carte Arduino et de la commande de l'afficheur :

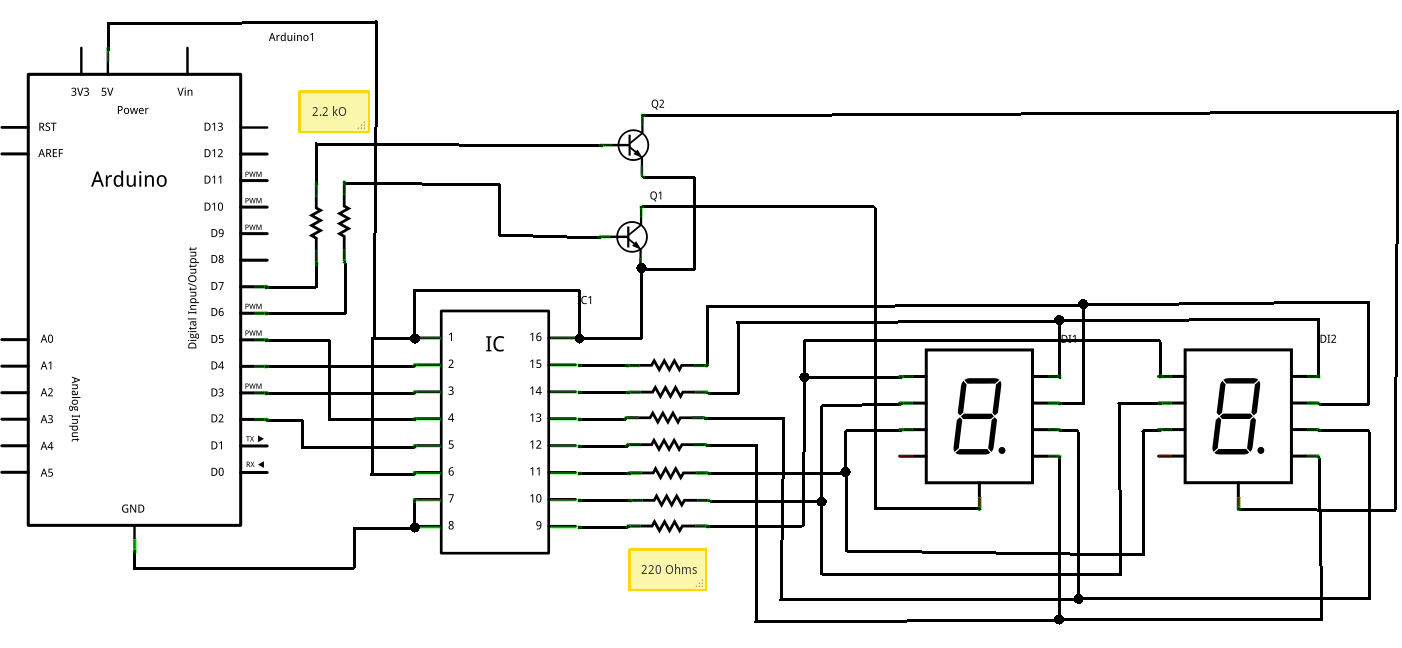
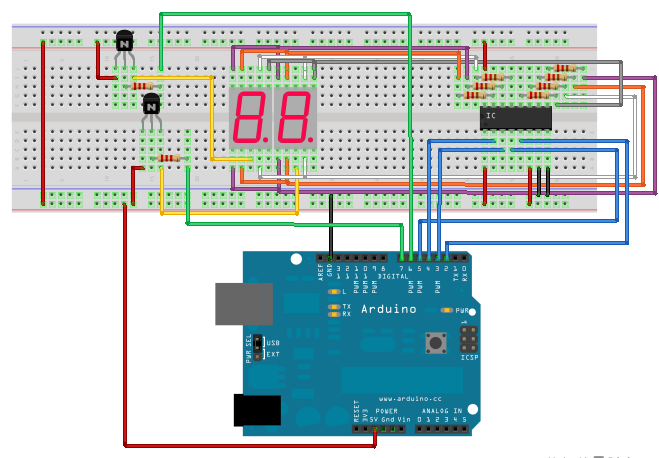


LA carte Arduino va soit mettre à la masse la base du transistor, soit la mettre à +5V. Dans le premier cas, il sera bloqué et l'afficheur sera éteint, dans le second il sera saturé et l'afficheur allumé.

Il en est de même pour chaque broche de l'afficheur. Elles seront au +5V ou à la masse selon la configuration que l'on aura définie dans le programme.

**Schéma final**

Et comme vous l’attendez surement depuis tout à l'heure, voici le schéma tant attendu (nous verrons juste après comment programmer ce nouveau montage) !

2\*7 segments schéma2\*7 segments breadboard

**Quelques détails techniques**

* Dans notre cas (et je vous passe les détails vraiment techniques et calculatoires), la résistance sur la base du transistor sera de 2.2k\Omega (si vous n'avez pas cette valeur, elle pourra être de 3.3k\Omega, ou encore de 3.9k\Omega, voir même de 4.7k\Omega).
* Les transistors seront des transistors bipolaires NPN de référence 2N2222, ou bien un équivalent qui est le BC547. Il en faudra deux donc.
* Le décodeur BCD est le même que précédemment (ou équivalent).

Et avec tout ça, on est prêt pour programmer ! :)

**...et de programmation**

Nous utilisons deux nouvelles broches servant à piloter chacun des interrupteurs (transistors). Chacune de ces broches doivent donc être déclarées en global (pour son numéro) puis régler comme sortie. Ensuite, il ne vous restera plus qu'à alimenter chacun des transistors au bon moment pour allumer l'afficheur souhaité. En synchronisant l'allumage avec la valeur envoyé au décodeur, vous afficherez les nombres souhaités comme bon vous semble. Voici un exemple de code complet, de la fonction setup() jusqu'à la fonction d'affichage. Ce code est commenté et vous ne devriez donc avoir aucun mal à le comprendre !

Ce programme est un compteur sur 2 segments, il compte donc de 0 à 99 et recommence au début dès qu'il a atteint 99. La vidéo se trouve juste après ce code.

//définition des broches du décodeur 7 segments (vous pouvez changer les numéros si bon vous semble)

const int bit\_A = 2;

const int bit\_B = 3;

const int bit\_C = 4;

const int bit\_D = 5;

//définitions des broches des transistors pour chaque afficheur (dizaines et unités)

const int alim\_dizaine = 6;

const int alim\_unite = 7;

void setup()

{

    //Les broches sont toutes des sorties

    pinMode(bit\_A, OUTPUT);

    pinMode(bit\_B, OUTPUT);

    pinMode(bit\_C, OUTPUT);

    pinMode(bit\_D, OUTPUT);

    pinMode(alim\_dizaine, OUTPUT);

    pinMode(alim\_unite, OUTPUT);

    //Les broches sont toutes mises à l'état bas

    digitalWrite(bit\_A, LOW);

    digitalWrite(bit\_B, LOW);

    digitalWrite(bit\_C, LOW);

    digitalWrite(bit\_D, LOW);

    digitalWrite(alim\_dizaine, LOW);

    digitalWrite(alim\_unite, LOW);

}

void loop() //fonction principale

{

    for(char i = 0; i<100; i++) //boucle qui permet de compter de 0 à 99 (= 100 valeurs)

    {

        afficher\_nombre(i); //appel de la fonction affichage avec envoi du nombre à afficher

    }

}

//fonction permettant d'afficher un nombre sur deux afficheurs

void afficher\_nombre(char nombre)

{

    long temps; //variable utilisée pour savoir le temps écoulé...

    char unite = 0, dizaine = 0; //variable pour chaque afficheur

    if(nombre > 9) //si le nombre reçu dépasse 9

        {

        dizaine = nombre / 10; //on récupère les dizaines

        }

    unite = nombre - (dizaine\*10); //on récupère les unités

    temps = millis(); //on récupère le temps courant

    // tant qu'on a pas affiché ce chiffre pendant au moins 500 millisecondes

        // permet donc de pouvoir lire le nombre affiché

    while((millis()-temps) < 500)

    {

        //on affiche le nombre

        //d'abord les dizaines pendant 10 ms

        digitalWrite(alim\_dizaine, HIGH); /\* le transistor de l'afficheur des dizaines est saturé,

                                                    donc l'afficheur est allumé \*/

        afficher(dizaine); //on appel la fonction qui permet d'afficher le chiffre dizaine

                digitalWrite(alim\_unite, LOW); // l'autre transistor est bloqué et l'afficheur éteint

                delay(10);

        //puis les unités pendant 10 ms

                digitalWrite(alim\_dizaine, LOW); //on éteint le transistor allumé

        afficher(unite); //on appel la fonction qui permet d'afficher le chiffre unité

        digitalWrite(alim\_unite, HIGH); //et on allume l'autre

                delay(10);

    }

}

//fonction écrivant sur un seul afficheur

//on utilise le même principe que vu plus haut

void afficher(char chiffre)

{

    if(chiffre >= 8)

    {

        digitalWrite(bit\_D, HIGH);

        chiffre = chiffre - 8;

    }

    if(chiffre >= 4)

    {

        digitalWrite(bit\_C, HIGH);

        chiffre = chiffre - 4;

    }

    if(chiffre >= 2)

    {

        digitalWrite(bit\_B, HIGH);

        chiffre = chiffre - 2;

    }

    if(chiffre >= 1)

    {

        digitalWrite(bit\_A, HIGH);

        chiffre = chiffre - 1;

    }

}

//le code est terminé

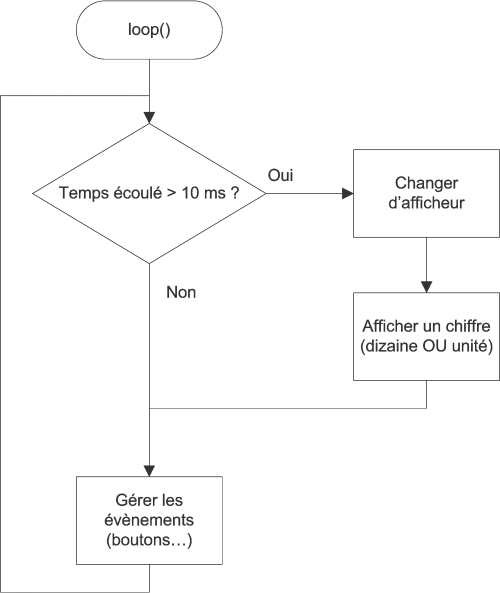
Voilà donc la vidéo présentant le résultat final :

<http://www.youtube.com/v/zgvV25s_ilQ>

**Contraintes des évènements**

Comme vous l'avez vu juste avant, afficher de manière alternative n'est pas trop difficile. Cependant, vous avez surement remarqué, nous avons utilisé des fonctions bloquantes (delay). Si jamais un évènement devait arriver pendant ce temps, nous aurions beaucoup de chance de le rater car il pourrait arriver "pendant" un délai d'attente pour l'affichage.

Pour parer à cela, je vais maintenant vous expliquer une autre méthode, préférable, pour faire de l'affichage. Elle s'appuiera sur l'utilisation de la fonction millis(), qui nous permettra de générer une boucle de rafraîchissement de l'affichage. Voici un organigramme qui explique le principe :



Comme vous pouvez le voir, il n'y a plus de fonction qui "attend". Tout se passe de manière continue, sans qu'il n'y ai jamais de pause. Ainsi, aucun évènement ne sera raté (en théorie, un évènement trèèèèèès rapide pourra toujours passer inaperçu).

Voici un exemple de programmation de la boucle principal (suivi de ses fonctions annexes) :

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

// --- setup() ---

void loop()

{

    //gestion du rafraichissement

    //si ça fait plus de 10 ms qu'on affiche, on change de 7 segments (alternance unité <-> dizaine)

    if((millis() - temps) > 10)

    {

        //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou dizaine)

        afficheur = !afficheur;

        //on affiche la valeur sur l'afficheur

        //afficheur : true->dizaines, false->unités

        afficher\_nombre(valeur, afficheur);

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

    }

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

}

//fonction permettant d'afficher un nombre

//elle affiche soit les dizaines soit les unités

void afficher\_nombre(char nombre, bool afficheur)

{

    char unite = 0, dizaine = 0;

    if(nombre > 9)

        dizaine = nombre / 10; //on recupere les dizaines

    unite = nombre - (dizaine\*10); //on recupere les unités

    //si "

    if(afficheur)

    {

        //on affiche les dizaines

        digitalWrite(alim\_unite, LOW);

        afficher(dizaine);

        digitalWrite(alim\_dizaine, HIGH);

    }

    else // égal à : else if(!afficheur)

    {

        //on affiche les unités

        digitalWrite(alim\_dizaine, LOW);

        afficher(unite);

        digitalWrite(alim\_unite, HIGH);

    }

}

//fonction écrivant sur un seul afficheur

void afficher(char chiffre)

{

    if(chiffre >= 8)

    {

        digitalWrite(bit\_D, HIGH);

        chiffre = chiffre - 8;

    }

    if(chiffre >= 4)

    {

        digitalWrite(bit\_C, HIGH);

        chiffre = chiffre - 4;

    }

    if(chiffre >= 2)

    {

        digitalWrite(bit\_B, HIGH);

        chiffre = chiffre - 2;

    }

    if(chiffre >= 1)

    {

        digitalWrite(bit\_A, HIGH);

        chiffre = chiffre - 1;

    }

}

Si vous voulez tester le phénomène de persistance rétinienne, vous pouvez changer le temps de la boucle de rafraichissement (ligne 9). Si vous l'augmenter, vous commencerez à vois les afficheurs clignoter. En mettant une valeur d'un peu moins de une seconde vous verrez les afficheurs s'illuminer l'un après l'autre.

Ce chapitre vous a appris à utiliser un nouveau moyen pour afficher des informations avec votre carte Arduino. L'afficheur peut sembler peu utilisé mais en fait de nombreuses applications existe ! (chronomètre, réveil, horloge, compteur de passage, afficheur de score, etc.). Par exemple, il pourra vous servir pour déboguer votre code et afficher la valeur des variables souhaitées...

# http://eskimon.fr/98-arduino-205-afficheurs-7-segments

## Première approche : côté électronique

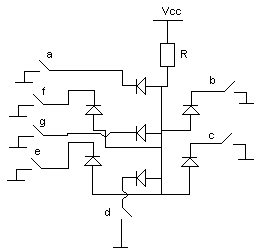
### Un peu (beaucoup) d’électronique

Comme son nom l’indique, l’afficheur 7 segments possède… 7 segments. Mais un segment c’est quoi au juste ? Et bien c’est une portion de l’afficheur, qui est allumée ou éteinte pour réaliser l’affichage. Cette portion n’est en fait rien d’autre qu’une LED qui au lieu d’être ronde comme d’habitude est plate et encastré dans un boiter. On dénombre donc 8 portions en comptant le point de l’afficheur (mais il ne compte pas en tant que segment à part entière car il n’est pas toujours présent). Regardez à quoi ça ressemble :

[ Arduino 205] Afficheurs 7 segmentsAfficheur 7 segments

#### Des LED, encore des LED

Et des LED, il y en a ! Entre 7 et 8 selon les modèles (c’est ce que je viens d’expliquer), voir beaucoup plus, mais on ne s’y attardera pas dessus. Voici un schéma vous présentant un modèle d’afficheur sans le point (qui au final est juste une LED supplémentaire rappelez-vous) :

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

Comme vous le voyez sur ce schéma, toutes les LED possèdent une broche commune, reliée entre elle. Selon que cette broche est la cathode ou l’anode on parlera d’afficheur à cathode commune ou… anode commune (vous suivez ?). Dans l’absolu, ils fonctionnent de la même façon, seule la manière de les brancher diffère (actif sur état bas ou sur état haut).

#### Cathode commune ou Anode commune

Dans le cas d’un afficheur à cathode commune, toutes les cathodes sont reliées entre elles en un seul point lui-même connecté à la masse. Ensuite, chaque anode de chaque segment sera reliée à une broche de signal. Pour allumer chaque segment, le signal devra être une tension positive. En effet, si le signal est à 0, il n’y a pas de différence de potentiel entre les deux broches de la LED et donc elle ne s’allumera pas ! Si nous sommes dans le cas d’une anode commune, les anodes de toutes les LED sont reliées entre elles en un seul point qui sera connecté à l’alimentation. Les cathodes elles seront reliées une par une aux broches de signal. En mettant une broche de signal à 0, le courant passera et le segment en question s’allumera. Si la broche de signal est à l’état haut, le potentiel est le même de chaque côté de la LED, donc elle est bloquée et ne s’allume pas ! Que l’afficheur soit à anode ou à cathode commune, on doit toujours prendre en compte qu’il faut ajouter une résistance de limitation de courant entre la broche isolée et la broche de signal. Traditionnellement, on prendra une résistance de 330 ohms pour une tension de +5V, mais cela se calcul (cf. chapitre 1, partie 2). Si vous voulez augmenter la luminosité, il suffit de diminuer cette valeur. Si au contraire vous voulez diminuer la luminosité, augmenter la résistance.

#### Choix de l’afficheur

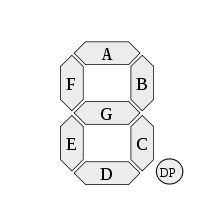
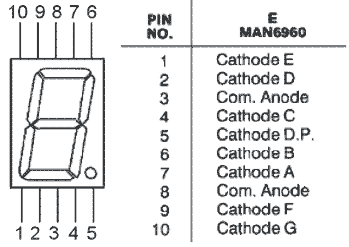
Pour la rédaction j’ai fait le choix d’utiliser des afficheurs à anode commune et ce n’est pas anodin. En effet et on l’a vu jusqu’à maintenant, on branche les LED du +5V vers la broche de la carte Arduino. Ainsi, dans le cas d’un afficheur à anode commune, les LED seront branchés d’un côté au +5V, et de l’autre côté aux broches de signaux. Ainsi, pour allumer un segment on mettra la broche de signal à 0 et on l’éteindra en mettant le signal à 1. On a toujours fait comme ça depuis le début, ça ne vous posera donc aucun problème. [ Arduino 205] Afficheurs 7 segments

### Branchement “complet” de l’afficheur

Nous allons maintenant voir comment brancher l’afficheur à anode commune.

#### Présentation du boîtier

Les afficheurs 7 segments se présentent sur un boîtier de type DIP 10. Le format DIP régie l’espacement entre les différentes broches du circuit intégré ainsi que d’autres contraintes (présence d’échangeur thermique etc…). Le chiffre 10 signifie qu’il possède 10 broches (5 de part et d’autre du boitier). Voici une représentation de ce dernier (à gauche) :



Voici la signification des différentes broches :

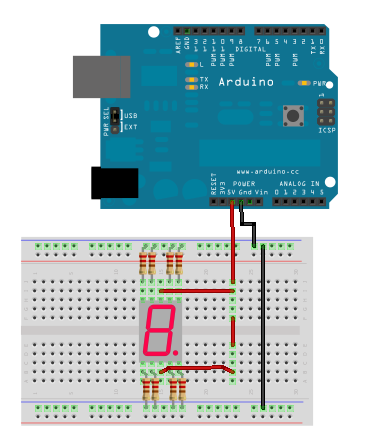
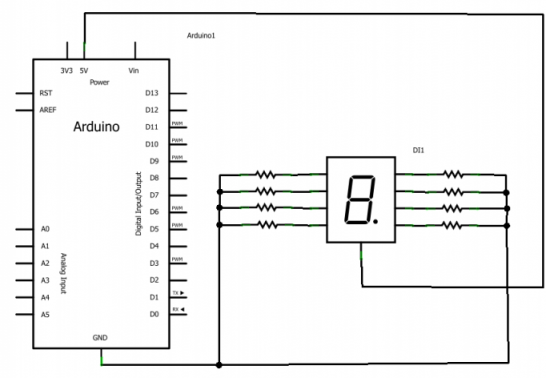
1. LED de la cathode E
2. LED de la cathode D
3. Anode commune des LED
4. LED de la cathode C
5. (facultatif) le point décimal.
6. LED de la cathode B
7. LED de la cathode A
8. Anode commune des LED
9. LED de la cathode F
10. LED de la cathode G

Pour allumer un segment c’est très simple, il suffit de le relier à la masse !

Nous cherchons à allumer les LED de l’afficheur, il est donc impératif de ne pas oubliez les résistances de limitations de courant !

#### Exemple

Pour commencer, vous allez tout d’abord mettre l’afficheur à cheval sur la plaque d’essai (breadboard). Ensuite, trouvez la broche représentant l’anode commune et reliez la à la future colonne du +5V. Prochaine étape, mettre une résistance de 330Ω sur chaque broche de signal. Enfin, reliez quelques une de ces résistances à la masse. Si tous se passe bien, les segments reliés à la masse via leur résistance doivent s’allumer lorsque vous alimentez le circuit. Voici un exemple de branchement :



Dans cet exemple de montage, vous verrez que tous les segment de l’afficheur s’allument ! Vous pouvez modifier le montage en déconnectant quelques unes des résistance de la masse et afficher de nombreux caractères.

Pensez à couper l’alimentation lorsque vous changer des fils de place. Les composants n’aiment pas forcément être (dé)branchés lorsqu’ils sont alimentés. Vous pourriez éventuellement leur causer des dommages.

#### Seulement 7 segments mais plein de caractère(s) !

Vous l’avez peut-être remarqué avec “l’exercice” précédent, un afficheurs 7 segments ne se limite pas à afficher juste des chiffres. Voici un tableau illustrant les caractères possibles et quels segments allumés. Attention, il est possible qu’il manque certains caractères !

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Caractère** | **seg. A** | **seg. B** | **seg. C** | **seg. D** | **seg. E** | **seg. F** | **seg. G** |
| 0 | x | x | x | x | x | x |  |
| 1 |  | x | x |  |  |  |  |
| 2 | x | x |  | x | x |  | x |
| 3 | x | x | x | x |  |  | x |
| 4 |  | x | x |  |  | x | x |
| 5 | x |  | x | x |  | x | x |
| 6 | x |  | x | x | x | x | x |
| 7 | x | x | x |  |  |  |  |
| 8 | x | x | x | x | x | x | x |
| 9 | x | x | x | x |  | x | x |
| A | x | x | x |  | x | x | x |
| b |  |  | x | x | x | x | x |
| C | x |  |  | x | x | x |  |
| d |  | x | x | x | x | x |  |
| E | x |  |  | x | x | x | x |
| F | x |  |  |  | x | x | x |
| H |  | x | x |  | x | x | x |
| I |  | x | x |  |  |  |  |
| J |  | x | x | x | x |  |  |
| L |  |  |  | x | x | x |  |
| o |  |  | x | x | x |  | x |
| P | x | x |  |  | x | x | x |
| S | x |  | x | x |  | x | x |
| t |  |  |  |  | x | x | x |
| U |  | x | x | x | x | x |  |
| y |  | x | x | x |  | x | x |
| ° | x | x |  |  |  | x | x |

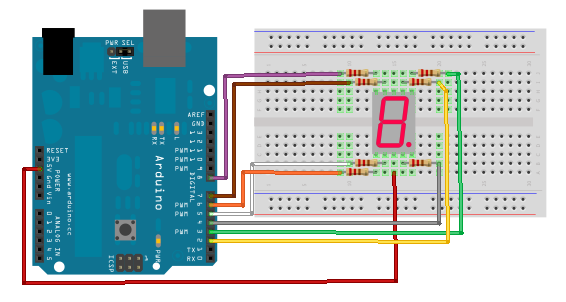
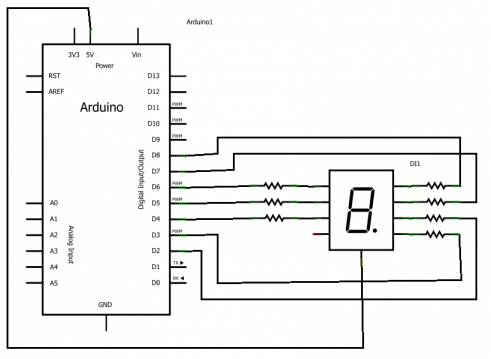
Aidez-vous de ce tableau lorsque vous aurez à coder l’affichage de caractères ! [ Arduino 205] Afficheurs 7 segments

## Afficher son premier chiffre !

Pour commencer, nous allons prendre en main un afficheur et lui faire s’afficher notre premier chiffre ! C’est assez simple et ne requiert qu’un programme très simple, mais un peu rébarbatif.

### Schéma de connexion

Je vais reprendre le schéma précédent, mais je vais connecter chaque broche de l’afficheur à une sortie de la carte Arduino. Comme ceci :



Vous voyez donc que chaque LED de l’afficheur va être commandée séparément les unes des autres. Il n’y a rien de plus à faire, si ce n’est qu’à programmer…

### Le programme

L’objectif du programme va être d’afficher un chiffre. Eh bien… c’est partit ! Quoi ?! Vous voulez de l’aide ? o\_O Ben je vous ai déjà tout dit y’a plus qu’à faire. En plus vous avez un tableau avec lequel vous pouvez vous aider pour afficher votre chiffre. Cherchez, je vous donnerais la solution ensuite. [spoiler id="" title="Réponse"] Solution :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | /\* On assigne chaque LED à une broche de l'arduino \*/  const int A = 2;  const int B = 3;  const int C = 4;  const int D = 5;  const int E = 6;  const int F = 7;  const int G = 8;  //notez que l'on ne gère pas l'affichage du point, mais vous pouvez le rajouter si cela vous chante    void setup()  {      //définition des broches en sortie      pinMode(A, OUTPUT);      pinMode(B, OUTPUT);      pinMode(C, OUTPUT);      pinMode(D, OUTPUT);      pinMode(E, OUTPUT);      pinMode(F, OUTPUT);      pinMode(G, OUTPUT);        //mise à l'état HAUT de ces sorties pour éteindre les LED de l'afficheur      digitalWrite(A, HIGH);      digitalWrite(B, HIGH);      digitalWrite(C, HIGH);      digitalWrite(D, HIGH);      digitalWrite(E, HIGH);      digitalWrite(F, HIGH);      digitalWrite(G, HIGH);  }    void loop()  {      //affichage du chiffre 5, d'après le tableau précédent      digitalWrite(A, LOW);      digitalWrite(B, HIGH);      digitalWrite(C, LOW);      digitalWrite(D, LOW);      digitalWrite(E, HIGH);      digitalWrite(F, LOW);      digitalWrite(G, LOW);  } |

Vous le voyez par vous-même, c’est un code hyper simple. Essayez de le bidouiller pour afficher des messages, par exemple, en utilisant les fonctions introduisant le temps. Ou bien compléter ce code pour afficher tous les chiffres, en fonction d’une variable définie au départ (ex: var = 1, affiche le chiffre 1 ; etc.). [/spoiler]

## Techniques d’affichage

Vous vous en doutez peut-être, lorsque l’on veut utiliser plusieurs afficheur il va nous falloir beaucoup de broches. Imaginons, nous voulons afficher un nombre entre 0 et 99, il nous faudra utiliser deux afficheurs avec 2∗7=14 broches connectées sur la carte Arduino. Rappel : une carte Arduino UNO possède… 14 broches entrées/sorties classiques. Si on ne fais rien d’autre que d’utiliser les afficheurs, cela ne nous gène pas, cependant, il est fort probable que vous serez amener à utiliser d’autres entrées avec votre carte Arduino. Mais si on ne libère pas de place vous serez embêté. Nous allons donc voir deux techniques qui, une fois cumulées, vont nous permettre d’utiliser seulement 4 broches pour obtenir le même résultat qu’avec 14 broches !

### Les décodeurs “4 bits -> 7 segments”

La première technique que nous allons utiliser met en œuvre un circuit intégré. Vous vous souvenez quand je vous ai parlé de ces bêtes là ? Oui, c’est le même type que le microcontrôleur de la carte Arduino. Cependant, le circuit que nous allons utiliser ne fait pas autant de choses que celui sur votre carte Arduino.

#### Décodeur BCD -> 7 segments

C’est le nom du circuit que nous allons utiliser. Son rôle est simple. Vous vous souvenez des conversions ? Pour passer du binaire au décimal ? Et bien c’est le moment de vous en servir, donc si vous ne vous rappelez plus de ça, allez revoir un peu [le cours](http://eskimon.fr/89-arduino-102-quelques-bases-elementaires#559277). Je disais donc que son rôle est simple. Et vous le constaterez par vous même, il va s’agir de convertir du binaire codé sur 4 bits vers un “code” utilisé pour afficher les chiffres. Ce code correspond en quelque sorte au tableau précédemment évoqué.

#### Principe du décodeur

Sur un afficheur 7 segments, on peut représenter aisément les chiffres de 0 à 9 (et en insistant un peu les lettres de A à F). En informatique, pour représenter ces chiffres, il nous faut au maximum 4 bits. Comme vous êtes des experts et que vous avez bien lu la partie sur le binaire, vous n’avez pas de mal à le comprendre. (0000)2 fera (0)10 et (1111)2 fera (15)10 ou (F)16. Pour faire 9 par exemple on utilisera les bits 1001. En partant de se constat, des ingénieurs ont inventé un composant au doux nom de “décodeur” ou “driver” 7 segments. Il reçoit sur 4 broches les 4 bits de la valeur à afficher, et sur 7 autres broches ils pilotent les segments pour afficher ladite valeur. Ajouter à cela une broche d’alimentation et une broche de masse on obtient 13 broches ! Et ce n’est pas fini. La plupart des circuits intégrés de type décodeur possède aussi une broche d’activation et une broche pour tester si tous les segments fonctionnent.

#### Choix du décodeur

Nous allons utiliser le composant nommé MC14543B comme exemple. Tout d’abord, ouvrez ce lien dans un nouvel onglet, il vous menera directement vers le pdf du décodeur :

[Datasheet du MC14543B](http://www.datasheetcatalog.org/datasheet2/4/09lwz6g28frlr15ayl6w0srxwz7y.pdf)

Les datasheets se composent souvent de la même manière. On trouve tout d’abord un résumé des fonctions du produit puis un schéma de son boîtier. Dans notre cas, on voit qu’il est monté sur un DIP 16 (DIP : Dual Inline Package, en gros “boîtier avec deux lignes de broches”). Si l’on continue, on voit la **table de vérité** faisant le lien entre les signaux d’entrées (INPUT) et les sorties (OUTPUT). On voit ainsi plusieurs choses :

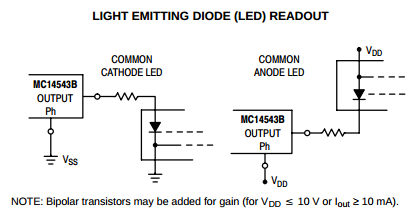
* Si l’on met la broche Bl (Blank, n°7) à un, toutes les sorties passent à zéro. En effet, comme son nom l’indique cette broche sert à effacer l’état de l’afficheur. Si vous ne voulez pas l’utiliser il faut donc la connecter à la masse pour la désactiver.
* Les entrées A, B, C et D (broches 5,3,2 et 4 respectivement) sont actives à l’état HAUT. Les sorties elles sont actives à l’état BAS (pour piloter un afficheur à anode commune) **OU** HAUT selon l’état de la broche PH (6). C’est là un gros avantage de ce composant, il peut inverser la logique de la sortie, le rendant alors compatible avec des afficheurs à anode commune (broche PH à l’état 1) ou cathode commune (Ph = 0)
* La broche BI (Blank Input, n°7) sers à inhiber les entrées. On ne s’en servira pas et donc on la mettra à l’état HAUT (+5V)
* LD (n°1) sert à faire une mémoire de l’état des sorties, on ne s’en servira pas ici
* Enfin, les deux broches d’alimentation sont la 8 (GND/VSS, masse) et la 16 (VCC, +5V)

N’oubliez pas de mettre des résistances de limitations de courant entre chaque segment et la broche de signal du circuit!

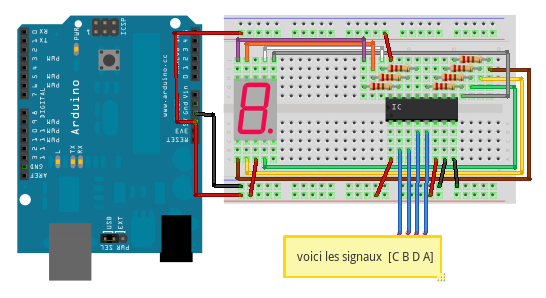
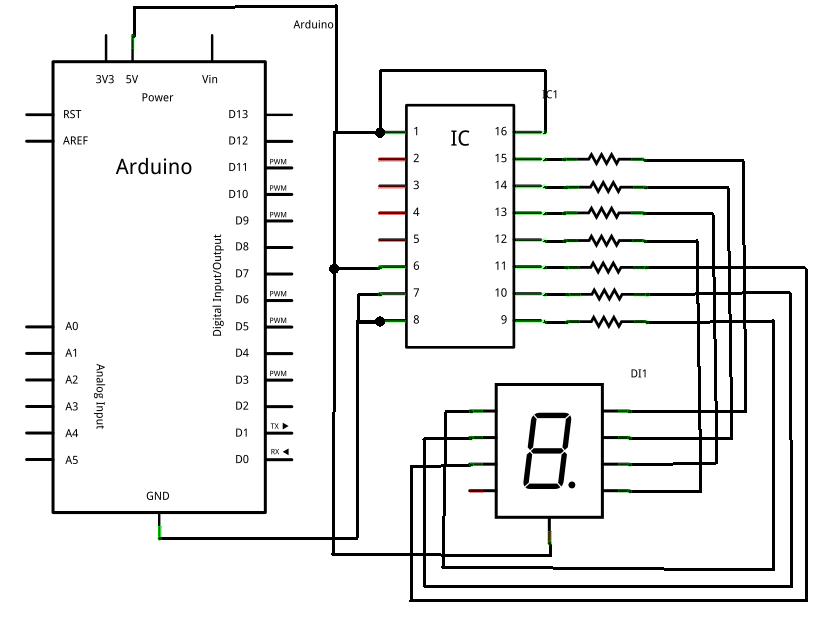
#### Fonctionnement

C’est bien beau tout ça mais comment je lui dis au décodeur d’afficher le chiffre 5 par exemple ?

Il suffit de regarder le datasheet et sa table de vérité (c’est le tableau avec les entrées et les sorties). Ce que reçoit le décodeur sur ses entrées (A, B, C et D) défini les états de ses broches de sortie (a,b,c,d,e,f et g). C’est tout ! Donc, on va donner un code binaire sur 4 bits à notre décodeur et en fonction de ce code, le décodeur affichera le caractère voulu. En plus le fabricant est sympa, il met à disposition des notes d’applications à la page 6 pour bien brancher le composant :



On voit alors qu’il suffit simplement de brancher la résistance entre le CI et les segments et s’assurer que PH à la bonne valeur et c’est tout ! En titre d’exercice afin de vous permettre de mieux comprendre, je vous propose de changer les états des entrées A, B, C et D du décodeur pour observer ce qu’il affiche. Après avoir réaliser votre schéma, regarder s’il correspond avec celui présent dans cette balise secrète. Cela vous évitera peut-être un mauvais branchement, qui sait ? [spoiler id="" title="Réponse"]



[/spoiler]

### L’affichage par alternance

La seconde technique est utilisée dans le cas où l’on veut faire un affichage avec plusieurs afficheurs. Elle utilise le phénomène de [persistance rétinienne](http://fr.wikipedia.org/wiki/Persistance_r%C3%A9tinienne). Pour faire simple, c’est grâce à cela que le cinéma vous parait fluide. On change une image toutes les 40 ms et votre œil n’a pas le temps de le voir, donc les images semble s’enchainer sans transition. Bref… Ici, la même stratégie sera utilisée. On va allumer un afficheur un certain temps, puis nous allumerons l’autre en éteignant le premier. Cette action est assez simple à réaliser, mais nécessite l’emploi de deux broche supplémentaires, de quatre autres composants et d’un peu de code. Nous l’étudierons un petit peu plus tard, lorsque nous saurons géré un afficheur seul.

## Utilisation du décodeur BCD

Nous y sommes, nous allons (enfin) utiliser la carte Arduino pour faire un affichage plus poussé qu’un unique afficheur. Pour cela, nous allons très simplement utiliser le montage précédent composé du décodeur BCD, de l’afficheur 7 segments et bien entendu des résistances de limitations de courant pour les LED de l’afficheur. Je vais vous montrer deux techniques qui peuvent être employées pour faire le programme.

#### Initialisation

Vous avez l’habitude maintenant, nous allons commencer par définir les différentes broches d’entrées/sorties. Pour débuter (et conformément au schéma), nous utiliserons seulement 4 broches, en sorties, correspondantes aux entrées du décodeur 7 segments. Voici le code pouvant traduire cette explication :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | const int bit\_A = 2;  const int bit\_B = 3;  const int bit\_C = 4;  const int bit\_D = 5;    void setup()  {      //on met les broches en sorties      pinMode(bit\_A, OUTPUT);      pinMode(bit\_B, OUTPUT);      pinMode(bit\_C, OUTPUT);      pinMode(bit\_D, OUTPUT);        //on commence par écrire le chiffre 0, donc toutes les sorites à l'état bas      digitalWrite(bit\_A, LOW);      digitalWrite(bit\_B, LOW);      digitalWrite(bit\_C, LOW);      digitalWrite(bit\_D, LOW);  } |

Ce code permet juste de déclarer les quatre broches à utiliser, puis les affectes en sorties. On les met ensuite toutes les quatre à zéro. Maintenant que l’afficheur est prêt, nous allons pouvoir commencer à afficher un chiffre !

#### Programme principal

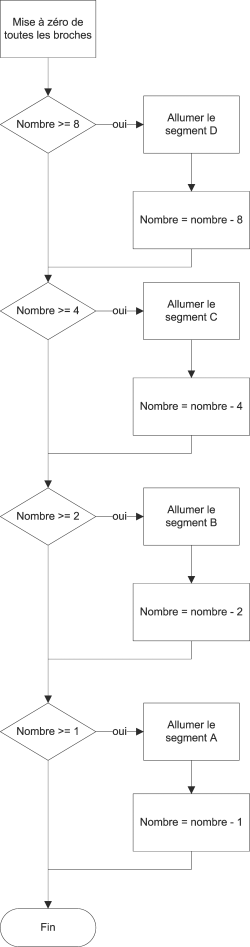
Si tout se passe bien, en ayant la boucle vide pour l’instant vous devriez voir un superbe 0 sur votre afficheur. Nous allons maintenant mettre en place un petit programme pour afficher les nombres de 0 à 9 en les incrémentant (à partir de 0) toutes les secondes. C’est donc un compteur. Pour cela, on va utiliser une boucle, qui comptera de 0 à 9. Dans cette boucle, on exécutera appellera la fonction afficher() qui s’occupera donc de l’affichage (belle démonstration de ce qui est une évidence [ Arduino 205] Afficheurs 7 segments[ Arduino 205] Afficheurs 7 segments).



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | void loop()  {      char i=0; //variable "compteur"      for(i=0; i<10; i++)      {          afficher(i); //on appel la fonction d'affichage          delay(1000); //on attend 1 seconde      }  } |

#### Fonction d’affichage

Nous touchons maintenant au but ! Il ne nous reste plus qu’à réaliser la fonction d’affichage pour pouvoir convertir notre variable en chiffre sur l’afficheur. Pour cela, il existe différentes solutions. Nous allons en voir ici une qui est assez simple à mettre en œuvre mais qui nécessite de bien être comprise. Dans cette méthode, on va faire des opérations mathématiques (tout de suite c’est moins drôle [ Arduino 205] Afficheurs 7 segments) successives pour déterminer quels bits mettre à l’état haut. Rappelez-vous, nous avons quatre broches à notre disposition, avec chacune un poids différent (8, 4, 2 et 1). En combinant ces différentes broches ont peu obtenir n’importe quel nombre de 0 à 15. Voici une démarche mathématique envisageable :



On peut coder cette méthode de manière assez simple et direct, en suivant cet organigramme :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | //fonction écrivant sur un seul afficheur  void afficher(char chiffre)  {      //on met à zéro tout les bits du décodeur      digitalWrite(bit\_A, LOW);      digitalWrite(bit\_B, LOW);      digitalWrite(bit\_C, LOW);      digitalWrite(bit\_D, LOW);        //On allume les bits nécessaires      if(chiffre >= 8)      {          digitalWrite(bit\_D, HIGH);          chiffre = chiffre - 8;      }      if(chiffre >= 4)      {          digitalWrite(bit\_C, HIGH);          chiffre = chiffre - 4;      }      if(chiffre >= 2)      {          digitalWrite(bit\_B, HIGH);          chiffre = chiffre - 2;      }      if(chiffre >= 1)      {          digitalWrite(bit\_A, HIGH);          chiffre = chiffre - 1;      }  } |

Quelques explications s’imposent… Le code gérant l’affichage réside sur les valeurs binaires des chiffres. Rappelons les valeurs binaires des chiffres :

|  |  |
| --- | --- |
| **Chiffre** | **DCBA** |
| 0 | (0000)2 |
| 1 | (0001)2 |
| 2 | (0010)2 |
| 3 | (0011)2 |
| 4 | (0100)2 |
| 5 | (0101)2 |
| 6 | (0110)2 |
| 7 | (0111)2 |
| 8 | (1000)2 |
| 9 | (1001)2 |

D’après ce tableau, si on veut le chiffre 8, on doit allumer le segment D, car 8 s’écrit (1000)2 ayant pour segment respectif DCBA. Soit D=1, C=0, B=0 et A=0. En suivant cette logique, on arrive à déterminer les entrées du décodeur qui sont à mettre à l’état HAUT ou BAS. D’une manière plus lourde, on aurait pu écrire un code ressemblant à ça :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | //fonction écrivant sur un seul afficheur  void afficher(char chiffre)  {      switch(chiffre)      {      case 0 :          digitalWrite(bit\_A, LOW);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, LOW);          break;      case 1 :          digitalWrite(bit\_A, HIGH);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, LOW);          break;      case 2 :          digitalWrite(bit\_A, LOW);          digitalWrite(bit\_B, HIGH);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, LOW);          break;      case 3 :          digitalWrite(bit\_A, HIGH);          digitalWrite(bit\_B, HIGH);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, LOW);          break;      case 4 :          digitalWrite(bit\_A, LOW);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, HIGH);          digitalWrite(bit\_D, LOW);          break;      case 5 :          digitalWrite(bit\_A, HIGH);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, HIGH);          digitalWrite(bit\_D, LOW);          break;      case 6 :          digitalWrite(bit\_A, LOW);          digitalWrite(bit\_B, HIGH);          digitalWrite(bit\_C, HIGH);          digitalWrite(bit\_D, LOW);          break;      case 7 :          digitalWrite(bit\_A, HIGH);          digitalWrite(bit\_B, HIGH);          digitalWrite(bit\_C, HIGH);          digitalWrite(bit\_D, LOW);          break;      case 8 :          digitalWrite(bit\_A, LOW);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, HIGH);          break;      case 9 :          digitalWrite(bit\_A, HIGH);          digitalWrite(bit\_B, LOW);          digitalWrite(bit\_C, LOW);          digitalWrite(bit\_D, HIGH);          break;      }  } |

Mais, c’est bien trop lourd à écrire. Enfin c’est vous qui voyez. [ Arduino 205] Afficheurs 7 segments

## Utiliser plusieurs afficheurs

Maintenant que nous avons affiché un chiffre sur un seul afficheur, nous allons pouvoir apprendre à en utiliser plusieurs (avec un minimum de composants en plus !). Comme expliqué précédemment, la méthode employée ici va reposer sur le principe de la persistance rétinienne, qui donnera l’impression que les deux afficheurs fonctionnent en même temps.

#### Problématique

Nous souhaiterions utiliser deux afficheurs, mais nous ne disposons que de seulement 6 broches sur notre Arduino, le reste des broches étant utilisé pour une autre application. Pour réduire le nombre de broches, on peut d’ores et déjà utilisé un décodeur BCD, ce qui nous ferait 4 broches par afficheurs, soit 8 broches au total. Bon, ce n’est toujours pas ce que l’on veut. Et si on connectait les deux afficheurs ensemble, en parallèle, sur les sorties du décodeur ? Oui mais dans ce cas, on ne pourrait pas afficher des chiffres différents sur chaque afficheur. Tout à l’heure, je vous ai parlé de commutation. Oui, la seule solution qui soit envisageable est d’allumer un afficheur et d’éteindre l’autre tout en les connectant ensemble sur le même décodeur. Ainsi un afficheur s’allume, il affiche le chiffre voulu, puis il s’éteint pour que l’autre puisse s’allumer à son tour. Cette opération est en fait un clignotement de chaque afficheur par alternance.

### Un peu d’électronique…

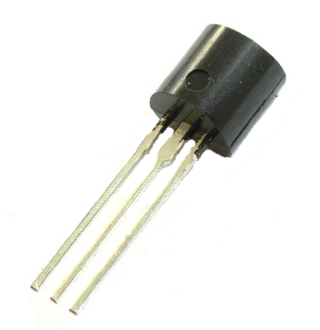
Pour faire commuter nos deux afficheurs, vous allez avoir besoin d’un nouveau composant, j’ai nommé : le **transistor** !

Transistor ? J’ai entendu dire qu’il y en avait plusieurs milliards dans nos ordinateurs ?

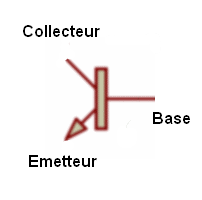
Et c’est tout à fait vrai. Des transistors, il en existe de différents types et pour différentes applications : amplification de courant/tension, commutation, etc. répartis dans plusieurs familles. Bon je ne vais pas faire trop de détails, si vous voulez en savoir plus, allez lire la première partie de ce chapitre (*lien à rajouter, en attente de la validation du chapitre en question*).

#### Le transistor bipolaire : présentation

Je le disais, je ne vais pas faire de détails. On va voir comment fonctionne un transistor bipolaire selon les besoins de notre application, à savoir, faire commuter les afficheurs. Un transistor, cela ressemble à ça :

Photo d’un transistor

Pour notre application, nous allons utiliser des **transistors bipolaires**. Je vais vous expliquer comment cela fonctionne. Déjà, vous pouvez observer qu’un transistor possède trois pattes. Cela n’est pas de la moindre importance, au contraire il s’agit là d’une chose essentielle ! En fait, le transistor bipolaire à une broche d’entrée (**collecteur**), une broche de sortie (**émetteur**) et une broche de commande (**base**). Son symbole est le suivant :



Ce symbole est celui d’un **transistor bipolaire de type NPN**. Il en existe qui sont de **type PNP**, mais ils sont beaucoup moins utilisés que les NPN. Quoi qu’il en soit, nous n’utiliserons que des transistors NPN dans ce chapitre.

#### Fonctionnement en commutation du transistor bipolaire

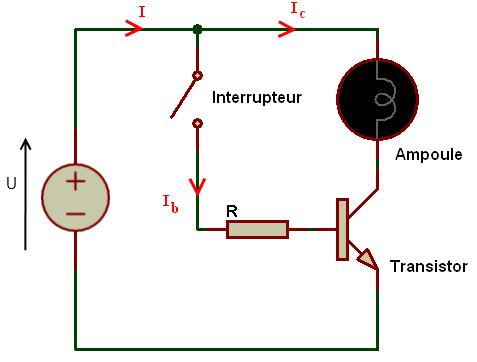
Pour faire simple, le transistor bipolaire NPN (c’est la dernière fois que je précise ce point) est un **interrupteur commandé en courant**.

Ceci est une présentation très vulgarisée et simplifiée sur le transistor pour l’utilisation que nous en ferons ici. Les usages et possibilités des transistors sont très nombreux et ils mériteraient un big-tuto à eux seuls ! Si vous voulez plus d’informations, rendez-vous sur le cours sur l’électronique ou approfondissez en cherchant des tutoriels sur le web. [ Arduino 205] Afficheurs 7 segments

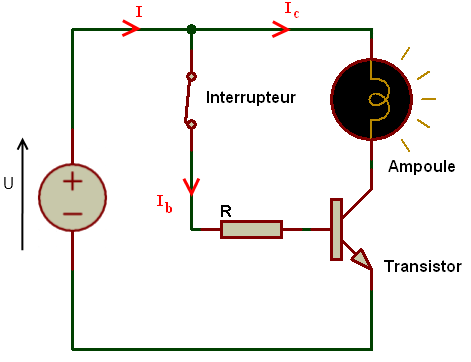
C’est tout ce qu’il faut savoir, pour ce qui est du fonctionnement. Après, on va voir ensemble comment l’utiliser et sans le faire griller ! [ Arduino 205] Afficheurs 7 segments

#### Utilisation générale

On peut utiliser notre transistor de deux manières différentes (pour notre application toujours, mais on peut bien évidemment utiliser le transistor avec beaucoup plus de flexibilités). A commencer par le câblage :

Câblage du transistor en commutation

Dans le cas présent, le collecteur (qui est l’entrée du transistor) se trouve être après l’ampoule, elle-même connectée à l’alimentation. L’émetteur (broche où il y a la flèche) est relié à la masse du montage. Cette disposition est “universelle”, on ne peut pas inverser le sens de ces broches et mettre le collecteur à la place de l’émetteur et vice versa. Sans quoi, le montage ne fonctionnerait pas. Pour le moment, l’ampoule est éteinte car le transistor ne conduit pas. On dit qu’il est **bloqué** et empêche donc le courant *IC* de circuler à travers l’ampoule. Soit *IC*=0 car *IB*=0. A présent, appuyons sur l’interrupteur :

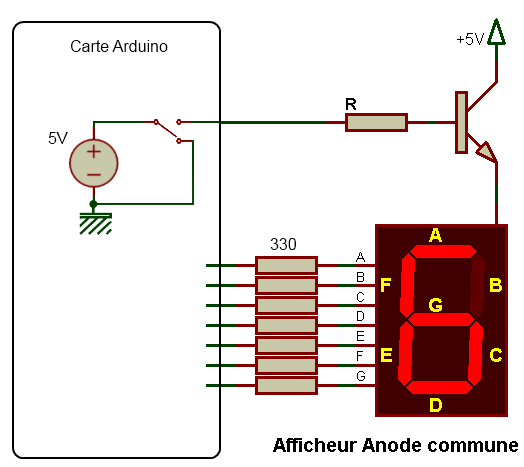
L’ampoule est allumée

Que se passe-t-il ? Eh bien la base du transistor, qui était jusqu’à présent “en l’air”, est parcourue par un courant électrique. Cette cause à pour conséquence de rendre le transistor **passant** ou **saturé** et permet au courant de s’établir à travers l’ampoule. Soit *IC*≠0 car *IB*≠0.

La résistance sur la base du transistor permet de le protéger des courants trop forts. Plus la résistance est de faible valeur, plus l’ampoule sera lumineuse. A l’inverse, une résistance trop forte sur la base du transistor pourra l’empêcher de conduire et de faire s’allumer l’ampoule. Rassurez\_vous, je vous donnerais les valeurs de résistances à utiliser. [ Arduino 205] Afficheurs 7 segments

#### Utilisation avec nos afficheurs

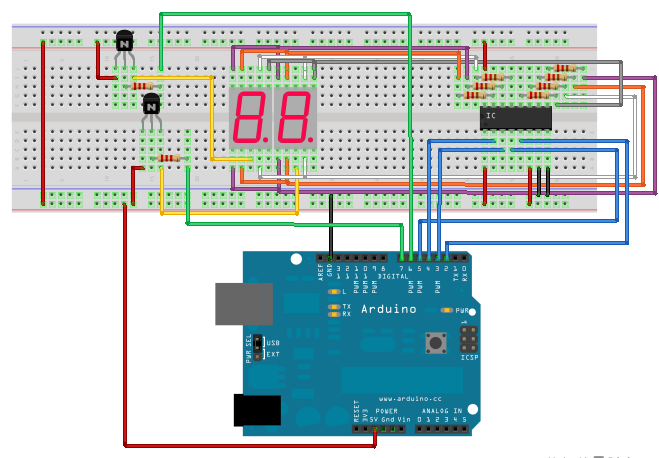
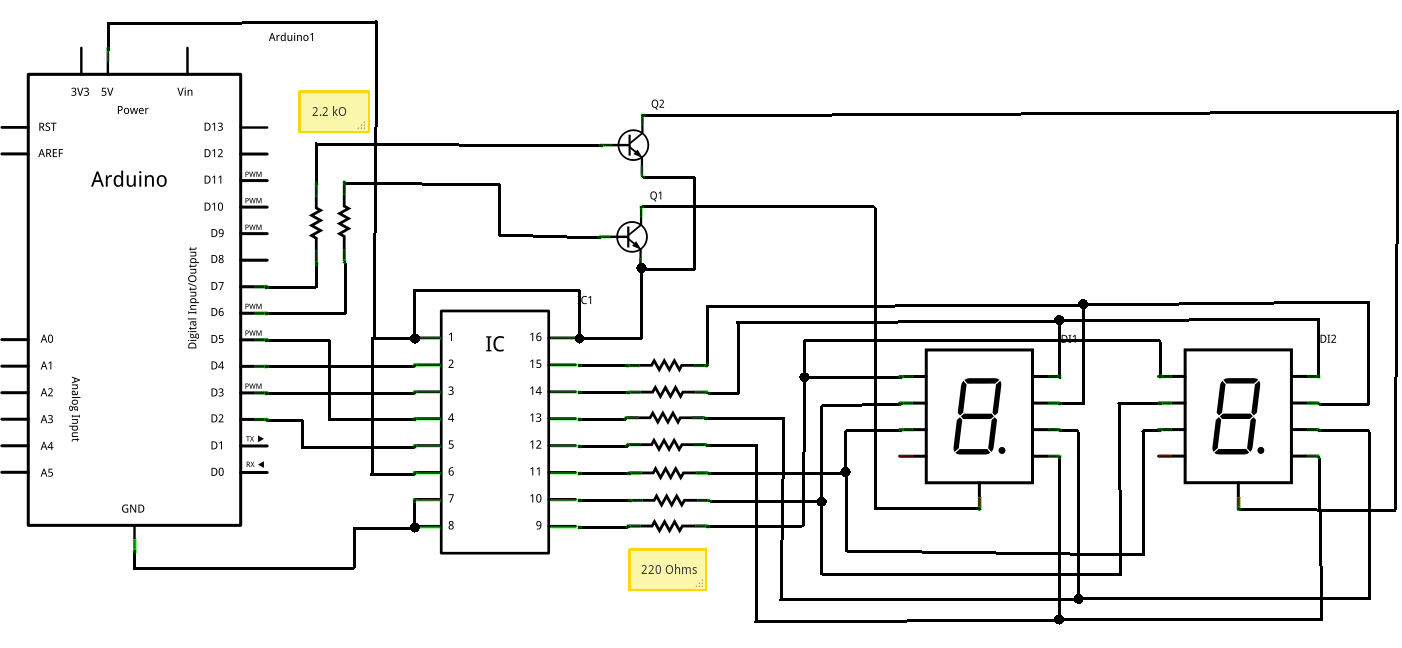
Voyons un peu comment on va pouvoir utiliser ce transistor avec notre Arduino. La carte Arduino est en fait le générateur de tension (schéma précédent) du montage. Elle va définir si sa sortie est de 0V (transistor bloqué) ou de 5V (transistor saturé). Ainsi, on va pouvoir allumer ou éteindre les afficheurs. Voilà le modèle équivalent de la carte Arduino et de la commande de l’afficheur :



LA carte Arduino va soit mettre à la masse la base du transistor, soit la mettre à +5V. Dans le premier cas, il sera bloqué et l’afficheur sera éteint, dans le second il sera saturé et l’afficheur allumé. Il en est de même pour chaque broche de l’afficheur. Elles seront au +5V ou à la masse selon la configuration que l’on aura définie dans le programme.

#### Schéma final

Et comme vous l’attendez surement depuis tout à l’heure, voici le schéma tant attendu (nous verrons juste après comment programmer ce nouveau montage) !



#### Quelques détails techniques

* Dans notre cas (et je vous passe les détails vraiment techniques et calculatoires), la résistance sur la base du transistor sera de 2.2*k*Ω (si vous n’avez pas cette valeur, elle pourra être de 3.3*k*Ω, ou encore de 3.9*k*Ω, voir même de 4.7*k*Ω).
* Les transistors seront des transistors bipolaires NPN de référence 2N2222, ou bien un équivalent qui est le BC547. Il en faudra deux donc.
* Le décodeur BCD est le même que précédemment (ou équivalent).

Et avec tout ça, on est prêt pour programmer ! [ Arduino 205] Afficheurs 7 segments

### …et de programmation

Nous utilisons deux nouvelles broches servant à piloter chacun des interrupteurs (transistors). Chacune de ces broches doivent donc être déclarées en global (pour son numéro) puis régler comme sortie. Ensuite, il ne vous restera plus qu’à alimenter chacun des transistors au bon moment pour allumer l’afficheur souhaité. En synchronisant l’allumage avec la valeur envoyé au décodeur, vous afficherez les nombres souhaités comme bon vous semble. Voici un exemple de code complet, de la fonction setup() jusqu’à la fonction d’affichage. Ce code est commenté et vous ne devriez donc avoir aucun mal à le comprendre ! Ce programme est un compteur sur 2 segments, il compte donc de 0 à 99 et recommence au début dès qu’il a atteint 99. La vidéo se trouve juste après ce code.

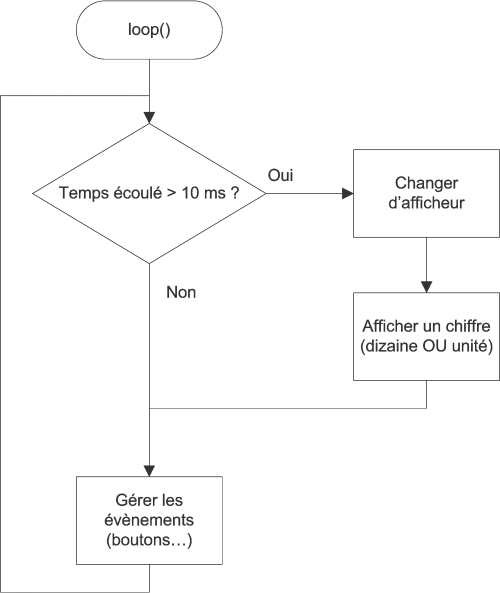


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105 | //définition des broches du décodeur 7 segments (vous pouvez changer les numéros si bon vous semble)  const int bit\_A = 2;  const int bit\_B = 3;  const int bit\_C = 4;  const int bit\_D = 5;    //définitions des broches des transistors pour chaque afficheur (dizaines et unités)  const int alim\_dizaine = 6;  const int alim\_unite = 7;    void setup()  {      //Les broches sont toutes des sorties      pinMode(bit\_A, OUTPUT);      pinMode(bit\_B, OUTPUT);      pinMode(bit\_C, OUTPUT);      pinMode(bit\_D, OUTPUT);      pinMode(alim\_dizaine, OUTPUT);      pinMode(alim\_unite, OUTPUT);        //Les broches sont toutes mises à l'état bas      digitalWrite(bit\_A, LOW);      digitalWrite(bit\_B, LOW);      digitalWrite(bit\_C, LOW);      digitalWrite(bit\_D, LOW);      digitalWrite(alim\_dizaine, LOW);      digitalWrite(alim\_unite, LOW);  }    void loop() //fonction principale  {      for(char i = 0; i<100; i++) //boucle qui permet de compter de 0 à 99 (= 100 valeurs)      {          afficher\_nombre(i); //appel de la fonction affichage avec envoi du nombre à afficher      }  }    //fonction permettant d'afficher un nombre sur deux afficheurs  void afficher\_nombre(char nombre)  {      long temps; //variable utilisée pour savoir le temps écoulé...      char unite = 0, dizaine = 0; //variable pour chaque afficheur        if(nombre > 9) //si le nombre reçu dépasse 9      {          dizaine = nombre / 10; //on récupère les dizaines      }        unite = nombre - (dizaine\*10); //on récupère les unités        temps = millis(); //on récupère le temps courant        // tant qu'on a pas affiché ce chiffre pendant au moins 500 millisecondes      // permet donc de pouvoir lire le nombre affiché      while((millis()-temps) < 500)      {          //on affiche le nombre            //d'abord les dizaines pendant 10 ms          digitalWrite(alim\_dizaine, HIGH); /\* le transistor de l'afficheur des dizaines est saturé,  donc l'afficheur est allumé \*/          afficher(dizaine); //on appel la fonction qui permet d'afficher le chiffre dizaine          digitalWrite(alim\_unite, LOW); // l'autre transistor est bloqué et l'afficheur éteint          delay(10);            //puis les unités pendant 10 ms          digitalWrite(alim\_dizaine, LOW); //on éteint le transistor allumé          afficher(unite); //on appel la fonction qui permet d'afficher le chiffre unité          digitalWrite(alim\_unite, HIGH); //et on allume l'autre          delay(10);      }  }    //fonction écrivant sur un seul afficheur  //on utilise le même principe que vu plus haut  void afficher(char chiffre)  {      digitalWrite(bit\_A, LOW);      digitalWrite(bit\_B, LOW);      digitalWrite(bit\_C, LOW);      digitalWrite(bit\_D, LOW);        if(chiffre >= 8)      {          digitalWrite(bit\_D, HIGH);          chiffre = chiffre - 8;      }      if(chiffre >= 4)      {          digitalWrite(bit\_C, HIGH);          chiffre = chiffre - 4;      }      if(chiffre >= 2)      {          digitalWrite(bit\_B, HIGH);          chiffre = chiffre - 2;      }      if(chiffre >= 1)      {          digitalWrite(bit\_A, HIGH);          chiffre = chiffre - 1;      }  }    //le code est terminé ! |

Voilà donc la vidéo présentant le résultat final :

## Contraintes des évènements

Comme vous l’avez vu juste avant, afficher de manière alternative n’est pas trop difficile. Cependant, vous avez surement remarqué, nous avons utilisé des fonctions bloquantes (delay). Si jamais un évènement devait arriver pendant ce temps, nous aurions beaucoup de chance de le rater car il pourrait arriver “pendant” un délai d’attente pour l’affichage. Pour parer à cela, je vais maintenant vous expliquer une autre méthode, préférable, pour faire de l’affichage. Elle s’appuiera sur l’utilisation de la fonction millis(), qui nous permettra de générer une boucle de rafraîchissement de l’affichage. Voici un organigramme qui explique le principe :



Comme vous pouvez le voir, il n’y a plus de fonction qui “attend”. Tout se passe de manière continue, sans qu’il n’y ai jamais de pause. Ainsi, aucun évènement ne sera raté (en théorie, un évènement trèèèèèès rapide pourra toujours passer inaperçu). Voici un exemple de programmation de la boucle principal (suivi de ses fonctions annexes) :



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76 | bool afficheur = false; //variable pour le choix de l'afficheur    // --- setup() ---    void loop()  {      //gestion du rafraichissement      //si ça fait plus de 10 ms qu'on affiche, on change de 7 segments (alternance unité <-> dizaine)      if((millis() - temps) > 10)      {          //on inverse la valeur de "afficheur" pour changer d'afficheur (unité ou dizaine)          afficheur = !afficheur;          //on affiche la valeur sur l'afficheur          //afficheur : true->dizaines, false->unités          afficher\_nombre(valeur, afficheur);          temps = millis(); //on met à jour le temps      }        //ici, on peut traiter les évènements (bouton...)  }    //fonction permettant d'afficher un nombre  //elle affiche soit les dizaines soit les unités  void afficher\_nombre(char nombre, bool afficheur)  {      char unite = 0, dizaine = 0;      if(nombre > 9)          dizaine = nombre / 10; //on recupere les dizaines      unite = nombre - (dizaine\*10); //on recupere les unités        //si "      if(afficheur)      {          //on affiche les dizaines          digitalWrite(alim\_unite, LOW);          afficher(dizaine);          digitalWrite(alim\_dizaine, HIGH);      }      else // égal à : else if(!afficheur)      {          //on affiche les unités          digitalWrite(alim\_dizaine, LOW);          afficher(unite);          digitalWrite(alim\_unite, HIGH);      }  }    //fonction écrivant sur un seul afficheur  void afficher(char chiffre)  {      digitalWrite(bit\_A, LOW);      digitalWrite(bit\_B, LOW);      digitalWrite(bit\_C, LOW);      digitalWrite(bit\_D, LOW);        if(chiffre >= 8)      {          digitalWrite(bit\_D, HIGH);          chiffre = chiffre - 8;      }      if(chiffre >= 4)      {          digitalWrite(bit\_C, HIGH);          chiffre = chiffre - 4;      }      if(chiffre >= 2)      {          digitalWrite(bit\_B, HIGH);          chiffre = chiffre - 2;      }      if(chiffre >= 1)      {          digitalWrite(bit\_A, HIGH);          chiffre = chiffre - 1;      }  } |

Si vous voulez tester le phénomène de persistance rétinienne, vous pouvez changer le temps de la boucle de rafraichissement (ligne 9). Si vous l’augmenter, vous commencerez à vois les afficheurs clignoter. En mettant une valeur d’un peu moins de une seconde vous verrez les afficheurs s’illuminer l’un après l’autre.

Ce chapitre vous a appris à utiliser un nouveau moyen pour afficher des informations avec votre carte Arduino. L’afficheur peut sembler peu utilisé mais en fait de nombreuses applications existe ! (chronomètre, réveil, horloge, compteur de passage, afficheur de score, etc.). Par exemple, il pourra vous servir pour déboguer votre code et afficher la valeur des variables souhaitées…

…/…

/\*Programme qui fait un compte à rebours eb utilisant un afficheur 7 segments avec un décodeur 7446.

Matériels :

- 1 carte Arduino UNO R3.

- 1 décodeur DM7446.

- 1 afficheur DIP 10, 7 segments à cathodes commune.

- 4 Résistances de 330 Ohm

\*/

/\*La broche 7 "A" du décodeur DM7446 est connectée à la broche 1 de l'Arduino\*/

/\*La broche 1 "B" du décodeur DM7446 est connectée à la broche 2 de l'Arduino\*/

/\*La broche 2 "C" du décodeur DM7446 est connectée à la broche 3 de l'Arduino\*/

/\*La broche 6 "D" du décodeur DM7446 est connectée à la broche 4 de l'Arduino\*/

const int bitsABCD[4]={1,2,3,4};

/\*tableau pour stocker le digit en binaire\*/

int leBit[4];

void setup()

{

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

pinMode(bitsABCD[iBit],OUTPUT);

for (int iBit = 0 ; iBit =0 ; chiffre--)

{

binFromDec(chiffre);

/\*parcours du tableau qui contient la conversion binaire

du chiffre.\*/

for (int iBit = 3 ; iBit >=0 ; iBit--)

digitalWrite(bitsABCD[iBit],leBit[iBit]);

delay(1000);

}

/\*sortir du loop()\*/

exit(0);

}

/\*convserion d'un chiffre sur 4 bits\*/

void binFromDec(int chiffre)

{

leBit[3] = 0;leBit[2] = 0;leBit[1] = 0;leBit[0] = 0;

if (chiffre & 8) leBit[3] = 1;

if (chiffre & 4) leBit[2] = 1;

if (chiffre & 2) leBit[1] = 1;

if (chiffre & 1) leBit[0] = 1;

}

…/…

Le code du 10/02/2014 n’est pas complet !!!  
Voilà le code complet :

/\*Programme qui fait un compte à rebours en utilisant un afficheur 7 segments.

Matériels :

- 1 carte Arduino UNO R3.

- 1 décodeur DM7446.

- 1 afficheur DIP 10, 7 segments à anode commune.

- 4 Résistances de 330 Ohm

\*/

/\*La broche 7 "A" du décodeur DM7446 est connectée à la broche 1 de l'Arduino\*/

/\*La broche 1 "B" du décodeur DM7446 est connectée à la broche 2 de l'Arduino\*/

/\*La broche 2 "C" du décodeur DM7446 est connectée à la broche 3 de l'Arduino\*/

/\*La broche 6 "D" du décodeur DM7446 est connectée à la broche 4 de l'Arduino\*/

const int bitsABCD[4]={1,2,3,4};

/\*tableau pour stocker le digit en binaire\*/

int leBit[4];

void setup()

{

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

pinMode(bitsABCD[iBit],OUTPUT);

for (int iBit = 0 ; iBit =0 ; chiffre--)

{

binFromDec(chiffre);

/\*parcours du tableau qui contient la conversion binaire

du chiffre.\*/

for (int iBit = 3 ; iBit >=0 ; iBit--)

digitalWrite(bitsABCD[iBit],leBit[iBit]);

delay(1000);

}

/\*sortir du loop()\*/

exit(0);

}

/\*convserion d'un chiffre sur 4 bits\*/

void binFromDec(int chiffre)

{

leBit[3] = 0;leBit[2] = 0;leBit[1] = 0;leBit[0] = 0;

if (chiffre & 8) leBit[3] = 1;

if (chiffre & 4) leBit[2] = 1;

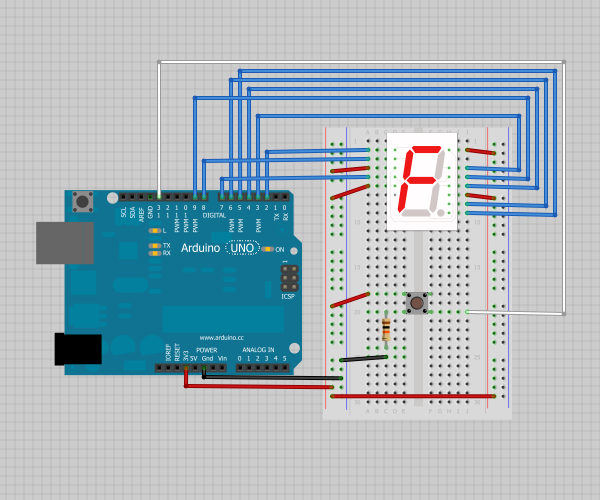
if (chiffre & 2) leBit[1] = 1;

if (chiffre & 1) leBit[0] = 1;

}

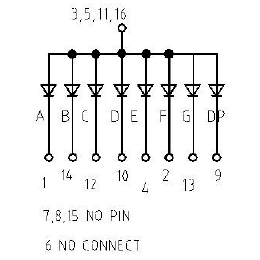
# http://forum.snootlab.com/viewtopic.php?f=38&t=914

Voici un exemple d'utilisation d'un afficheur 7 segments : un compteur d'impulsions.  
A chaque appui sur un bouton poussoir on incrémente la valeur du compteur et on envoie le résultat sur l'afficheur 7 segments. Rien de bien compliqué au niveau des données qui rentrent mais ça va nous permettre de nous concentrer sur l'afficheur ;)  
Tout d'abord, pour le brochage de l'afficheur, je vous invite à aller lire [la documentation disponible sur le forum](http://forum.snootlab.com/viewtopic.php?f=38&t=589), pour le reste, c'est en dessous !  
  
**- MATÉRIEL -**   
  
- [Arduino Uno](http://snootlab.com/arduino/142-arduino-uno-rev3.html) (x1)  
- [Afficheur 7 segments](http://snootlab.com/composants/263-afficheur-7-segments-vert.html) (x1)  
- [Bouton poussoir](http://snootlab.com/composants/102-bouton-poussoir-12mm.html) (x1)  
- [Résistance 10 kOhms](http://snootlab.com/composants/197-resistances-10-kohms-5-1-4w.html) (x1)  
  
**- SCHÉMA DU MONTAGE -**



compteur\_aff7seg\_schéma.png (55.01 Kio) Vu 1318 fois

Pour rappel :



pinout\_aff7seg.png (11.02 Kio) Vu 1318 fois

:!:Ici nous avons un afficheur à anode commune, si vous en avez un à cathode commune il faut modifier le montage : toutes les broches reliées au +5V doivent aller à la masse (GND) et il faut mettre des résistances de 150 Ohms entre les broches de l'Arduino et celles de votre afficheur. Il y a également une légère modification à apporter au code (voir ci-dessous). Vous aurez aussi remarqué que nous alimentons l'afficheur avec du 3.3V depuis l'Arduino : si vous souhaitez l'alimenter en 5V il vous faudra ajouter des résistances de 150 Ohms entre les pins de votre afficheur et celles de l'Arduino !  
  
**- CODE -**

Code: [Tout sélectionner](http://forum.snootlab.com/viewtopic.php?f=38&t=914)

/\*  
 \* Compteur d'impulsions modulo 10 avec afficheur 7 segments  
 \*/  
  
// Bouton poussoir relié à pin D13 de l'Arduino  
#define bouton 13  
  
// déclaration des constantes liées au bouton  
int compte = 0;  
int etatBouton = LOW;   
int etatPrecedent = LOW;  
  
// Définition des segments à allumer pour l'affichage des chiffres :  
// Seuls la barre centrale "G" et le point décimal sont éteints pour afficher 0, etc.  
const byte nombre[10] =   
{  
 //ABCDEFG°    
  B11111100, // 0  
  B01100000, // 1  
  B11011010, // 2  
  B11110010, // 3  
  B01100110, // 4  
  B10110110, // 5  
  B10111110, // 6  
  B11100000, // 7  
  B11111110, // 8  
  B11110110, // 9  
};  
  
// Tableau pour stocker les pins reliées aux segments (°, G, F, E, D, C, B, A)  
// segments[0] correspond au bit de poids faible (LSB), ci-dessus il s'agit du point décimal  
// donc on fait la déclaration dans l'ordre inverse  
const int segments[8] = {5, 9, 8, 7, 6, 4, 3, 2};  
  
void setup()  
{  
  pinMode(bouton, INPUT);  
    
  // optimisation du setup grâce à une boucle for pour éviter les multiples pinMode() et digitalWrite();  
  for (int i=0; i<8; i++)  
  {  
    pinMode(segments[i], OUTPUT);  
    digitalWrite(segments[i], HIGH);  
  }  
}  
  
void loop()  
{  
  etatBouton = digitalRead(bouton);  
  // si l'état du bouton a changé  
  if (etatBouton != etatPrecedent)  
  {  
    // et qu'il est haut (= on appuie dessus)  
    if (etatBouton == HIGH)  
    {  
      // on incrémente le compteur  
      compte++;  
      // sauf s'il a dépassé 9...  
      if (compte > 9)  
      {  
        // ...auquel cas on le réinitialise  
        compte = 0;  
      }  
    }  
  }  
  // on affiche le nombre souhaité  
  affichage(compte);  
  // et on stocke l'atat du bouton pour le prochain tour  
  etatPrecedent = etatBouton;  
}  
  
void affichage(int chiffre)  
{  
  // variable servant à stocker l'état des différents segments  
  boolean segState;  
    
  for (int seg = 0; seg < 8; seg++)  
  {  
    // lecture des bytes pour déterminer l'état des segments  
    segState = bitRead(nombre[chiffre], seg);  
    // inversion (afficheur à anode commune uniquement)  
    segState = !segState;  
    // affichage  
    digitalWrite(segments[seg], segState);  
  }  
}

**- EXPLICATIONS -**   
  
On cherche à savoir quels segments allumer pour chacun des chiffres. Or nous avons défini des bytes (un byte = un "mot" de 8 bits) correspondant à l'état des segments pour chacun des chiffres que l'on veut afficher.  
Nous utilisons donc la fonction bitRead(x, n), avec x = variable à étudier, n = quel bit lire. Cette fonction renvoie un "0" ou un "1" selon la valeur du bit sélectionné. En utilisant une boucle for(), on lit successivement tous les bits du premier (point décimal) au dernier (segment G). On peut donc remonter les différents états dans lesquels doivent être les différents segments, il ne nous reste plus qu'à les afficher !  
  
:!:On inverse ici l'état de la variable booléenne car on a un afficheur à anode commune (pins actives à l'état LOW), il vous faudra supprimer cette ligne si vous utilisez un afficheur à cathode commune.  
  
Note : le point décimal n'étant pas utilisé dans ce sketch, on aurait pu initialiser la boucle for() avec seg=1 pour ne pas le prendre en compte dans la lecture des bits, en économisant du temps de calcul ;)

# Cristaux Liquide / LCD

# http://www.pobot.org/LCD-sur-I2C-simplement.html

Le prix des afficheurs à cristaux liquides est en baisse, et il est de plus en plus fréquent de trouver des circuits de commande simples, évitant la gestion du protocole et économisant surtout le nombre de pattes d’entrée/sortie nécessaires. Georges utilise pour son projet un LCD sur bus I2C à base de PCF8574 dont voici la mise en œuvre.

  Sommaire

* [Le matériel](http://www.pobot.org/LCD-sur-I2C-simplement.html#le_materiel)
* [Le logiciel](http://www.pobot.org/LCD-sur-I2C-simplement.html#le_logiciel)

Il y autant de circuits de communication avec un écran LCD que de contrôleurs de LCD. Deux possibilités de se retrouver bloquer par un manque de documentation. La récupération de ce type de composant est donc souvent infructueuse, et on a alors recours à l’achat d’un écran du commerce.

Pour 15 euros (prix public en France), on peut désormais disposer d’un module I2C LCD avec un écran 2 lignes 16 colonnes, rétro-éclairage, fond bleu et caractères blanc. Il existe aussi des versions à 4 lignes et 20 colonnes (B2CQshop par exemple). Dans les deux cas, c’est une bonne affaire.

[Module I2C/TWI LCD chez Zartronic (FR)](http://www.zartronic.fr/module-i2ctwi-lcd1602-p-120.html)

[Ecran 4x20 avec module I2C chez B2cqShop (CN)](http://cgi.ebay.fr/ws/eBayISAPI.dll?ViewItem&item=150653960967&ssPageName=STRK:MEWNX:IT&_trksid=p3984.m1439.l2649#ht_3658wt_932)

[](http://www.pobot.org/IMG/jpg/lcd_I2C.jpg)

Cette baisse s’explique par un nouveau marché grâce à la carte à base de micro-contrôleur Arduino. Il n’est donc pas étonnant que nous aussi nous ayons choisi ce support pour tester cet écran.

Une fois les principes bien maitrisés, il sera toujours possible d’utiliser un autre support, à base de micro-contrôleur ou même d’un PC.

### Le matériel

L’écran LCD est équipé d’une puce très courante pour ce type d’afficheurs de texte, l’Hitachi HD44780 à port parallèle. Chacune des pattes est soudée à une petite carte avec une puce PCF8574 qui fait la conversion parallèle / I2C.

La puce PCF8574 est une simple extension de port pour 8 pattes d’entrée/sortie permettant de les contrôler une à une par I2C. Il y aura donc tout un traitement logiciel dans le micro-contrôleur maitre, contrairement à d’autres solutions qui définissent un protocole spécifique à l’affichage de texte, une puce sur l’esclave I2C faisant l’adaptation de protocole et toute la gestion du curseur ce qui allège la partie embarquée du maître. Un point à prendre en considération si vous craignez d’être à cours de ressources.

[Datasheet du PCF8574 chez NXP](http://www.nxp.com/documents/data_sheet/PCF8574.pdf)

Ce circuit additionnel ajoute de l’épaisseur mais reste très discret. Il est surtout pratique, avec un connecteur à 4 pattes (masse, alimentation 5 volts, signaux série de données (SDA) et d’horloge (SCL) du bus I2C, ainsi qu’un potentiomètre réglable pour le contraste.

On connecte la masse et l’alimentation au connecteur femelle d’une Arduino Uno, tandis que le signal SDA va sur la patte ANALOG 4 et le signal SCL va sur la patte ANALOG 5. Il s’agit en fait des ports TWI du micro-contrôleur AVR, car sur ce type de puce chaque patte peut avoir plusieurs rôles.

Il n’est pas possible de connecter ailleurs ces signaux, à moins de développer votre propre code logiciel de gestion du bus I2C, sans passer par le périphérique intégré TWI de la puce : vous perdrez en ressources libres pour votre programme principal et le résultat n’est pas garanti.

ATTENTION !

Pour une carte Arduino Mega (ou votre propre carte utilisant un autre micro-contrôleur qu’un ATmega8 ou 168 et 328 comme les Arduino classiques), il faut rechercher quelles pattes portent les signaux SDA et SCL du bus I2C/TWI.

[Correspondance des pattes de l’ATmega2560 avec Arduino](http://arduino.cc/en/Hacking/PinMapping2560)

Ce problème s’avère particulièrement sournois quand vous utilisez une extension (shield) qui comporte un connecteur I2C : ce dernier n’est alors plus d’aucune utilité pour votre carte Mega.

### Le logiciel

**L’adressage de l’esclave I²C**

L’adresse I2C du module est 0x27 (norme 7 bits) : il faut donc vous assurer qu’aucun autre module sur votre montage n’est connecté avec cette adresse. Mais de toute façon il est toujours conseillé de commencer ses tests avec seulement le maitre (votre micro-contrôleur ou votre carte Arduino) et un seul esclave connecté.

La datasheet du PCF8574 montre que trois pattes permettent de changer l’adresse. Dans le circuit que nous testons, elles sont toutes trois à 1 (ce qui complète la partie fixe pour donner 010 0111 soit 0x27).

Si vous n’utilisez pas une Arduino ou un dispositif logiciel similaire utilisant une adresse à 7 bits, pensez à ajouter le 0 ou le 1 selon que vous êtes en écriture ou en lecture : l’adresse 8 bits devient alors 0100 1110 soit 0x4E ou 0x4F.

**Logiciel fourni**

Le module LCD I2C est fourni avec une archive contenant une bibliothèque de code pour Arduino.

La bibliothèque s’appelle "Liquid Crystal I2C" et a été testée avec succès sur un environnement de développement Windows + Arduino 0021.

La liste de fonctions est presque compatible avec l’API LCD du Playground Arduino, une page collaborative qui rassemble toutes les façons standards de connecter un écran à cristaux liquides.

[Déclaration de l’API LCD](http://www.arduino.cc/playground/Code/LCDAPI)

On retrouve donc toutes les fonctions indispensables :

- le constructeur (LiquidCrystal\_I2C) avec adresse, colonnes et lignes  
- l’initialisation : init  
- l’éclairage : backlight, noBacklight  
- l’écriture : print  
- la gestion du curseur : setCursor

A noter que "println" ne fonctionne pas (ajout d’un caractère illisible au lieu d’un retour à la ligne et qu’il manque certaines fonctions). Mais rien n’empêche de les ajouter, puisque toute la gestion de l’écran est faite depuis la bibliothèque logicielle, la puce PCF8574 étant simplement un convertisseur d’entrées/sorties numériques.

Attention, une page du Playground parle des différentes bibliothèques pour LCD par I2C dont une Liquid Crystal I2C mais il semble que ce ne soit pas la même. Celle de B2cqShop et de DFRobot ne contient pas la fonction setDelay() annoncée obligatoire selon l’API par exemple.

Tous les fichiers constituants la bibliothèque (code, exemples) doivent être placés dans le répertoire "libraries" du répertoire du programme Arduino, pas dans votre répertoire "sketchbook" contenant vos codes. Normalement (en version 0021 en tout cas), ce répertoire "LiquidCrystal\_I2C" va se retrouver à côté d’un répertoire "LiquidCrystal" d’origine.

**Premier test**

Le code est simple :

#include

<Wire.h>

#include

<LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd(0x27,20,4) ;

void setup()   
 *lcd.init() ;   
lcd.cursor\_on() ;  
lcd.blink\_on() ;  
lcd.backlight() ;  
lcd.setCursor(0,0) ;  
lcd.print("Hello, world !") ;  
lcd.setCursor(4,2) ;  
lcd.print("Pobot is alive !") ;  
delay(1000) ;   
lcd.cursor\_off() ;  
lcd.blink\_off() ;*

void loop()

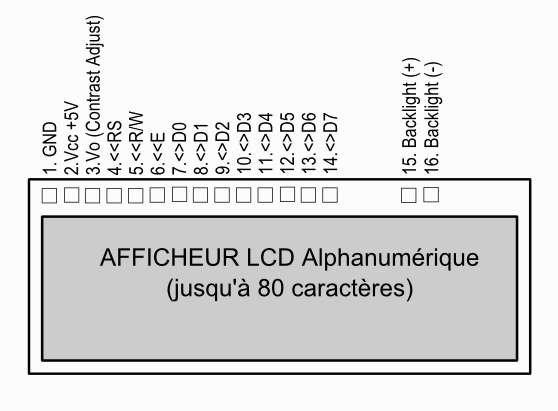
Ce code pèse 2836 octets, preuve que la bibliothèque prend quand même pas mal de place. Pour rappel, tout le code du robot de la Coupe de France 2006 tenait dans 6000 octets.

# <http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD>

## Préparation d'un afficheur LCD pour une utilisation simplifiée avec une carte Arduino, PIC, etc..

#### Brochage d'un afficheur LCD alphanumérique

Pour mémoire, le brochage d'un afficheur LCD alphanumérique standard est le suivant :

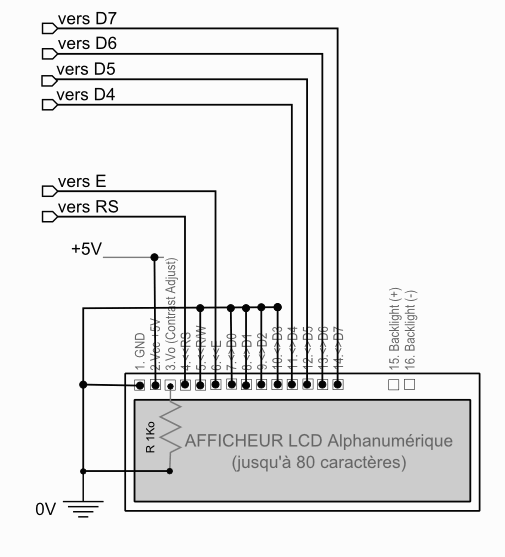


#### Connexion à réaliser pour une utilisation simplifiée :

Un afficheur LCD peut être utilisé en mode "4 bits" ou "8 bits". Dans le mode 4 bits, on utilise que 4 lignes de données pour envoyer les données à l'afficheur. C'est ce mode qui est le plus pratique. Les connexions à réaliser sont alors les suivantes :

* broches RS et E connectées à 2 broches numériques en sortie
* broches D4 à D7 connectées à 4 broches numériques en sortie
* Le + et - connectées au 5V et à la masse (0V)
* Une résistance de réglage du contraste entre le +5V et la broche Vo (en pratique 1Kohm)
* la broche RW et les broches D0 - D3 non utilisées et connectées à la masse.

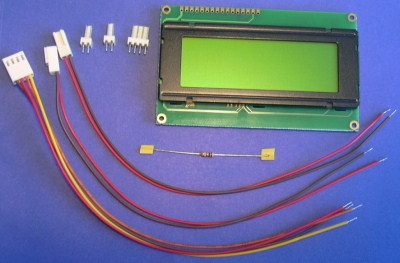
Voici le schéma de cette connexion simplifiée de l'afficheur LCD alpha-numérique :



#### Le matériel nécessaire

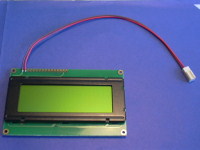
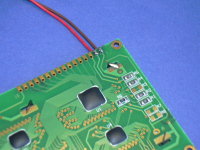
Pour préparer l'afficheur LCD, il faut :

* un afficheur LCD alphanumérique standard jusqu'au format 4x20.
* 2 connecteurs sur fils 2 broches + 2 connecteur droit 2 broches
* 1 connecteur sur fils 4 broches + 1 connecteur droit 4 broches
* 1 résistance 1kohms - 1/4w
* +/- un condensateur de 100nF

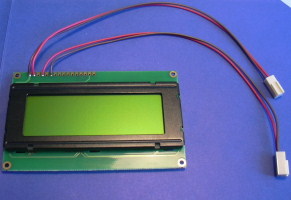
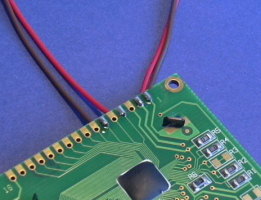


#### La préparation de l'afficheur LCD en photos

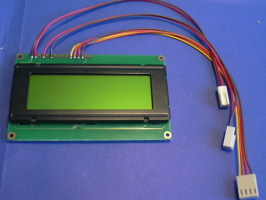
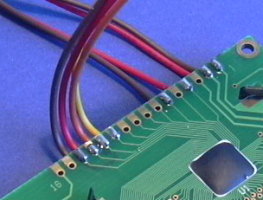
* On commence par souder le connecteur 2 broches sur fils sur le + et - de l'afficheur :



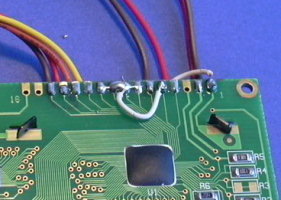
* On soude ensuite le connecteur 2 broches sur fils sur les broches RS et E :



* On soude ensuite le connecteur 4 broches sur fils sur les broches D4 à D7 :



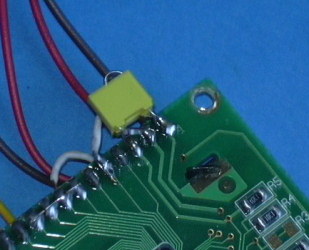
* On soude ensuite entre-elles les broches D0 à D3, la broche RW et la broche 0V



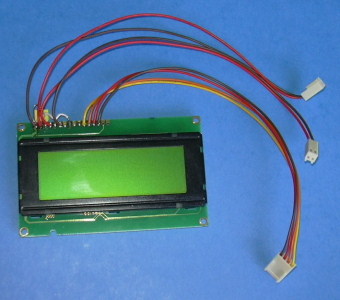
* On soude la résistance de 1Ko entre le 0V et la broche Vo



* On peut enfin souder un condensateur 100nF entre le + et le - (pas indispensable...)



* Voici à quoi ressemble le LCD préparé fini et prêt à l'emploi avec votre carte Arduino :



# http://www.mon-club-elec.fr/pmwiki\_mon\_club\_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte..

Afficher du texte sur un afficheur LCD alphanumérique

Par X. HINAULT - Créé le 03/02/2010

**Sur cette page...** ([Masquer](javascript:toggle('tocid');))

1. 1. [Présentation](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc1)
2. 2. [Matériel Nécessaire](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc2)
   1. 2.1 [L'espace de développement Arduino](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc3)
   2. 2.2 [Le matériel suivant pour réaliser le montage associé](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc4)
3. 3. [Instructions de montage](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc5)
4. 4. [Le schéma théorique du montage](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc6)
5. 5. [Le circuit du montage](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc7)
6. 6. [Au niveau de la partie déclarative :](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc8)
7. 7. [Au niveau de la fonction d'initialisation setup( ) :](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc9)
8. 8. [Au niveau de la boucle principale, la fonction loop ( ) :](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc10)
9. 9. [Mise en oeuvre du programme](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc11)
   1. 9.1 [Préparation du montage et programmation de la carte Arduino :](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc12)
   2. 9.2 [Fonctionnement](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc13)
10. 10. [Le programme complet en langage Arduino :](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte#toc14)

## 1.  Présentation

Ce programme montre comment afficher du texte sur un afficheur LCD alphanumérique.

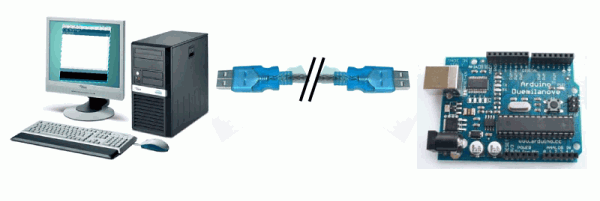
Ce programme utilise les fonctionnalités suivantes :

* un afficheur LCD alphanumérique 4x20 en mode 4 bits

## 2.  Matériel Nécessaire

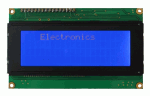
### 2.1  L'espace de développement Arduino

* ... pour éditer, compiler le programme et programmer la carte Arduino.



### 2.2  Le matériel suivant pour réaliser le montage associé

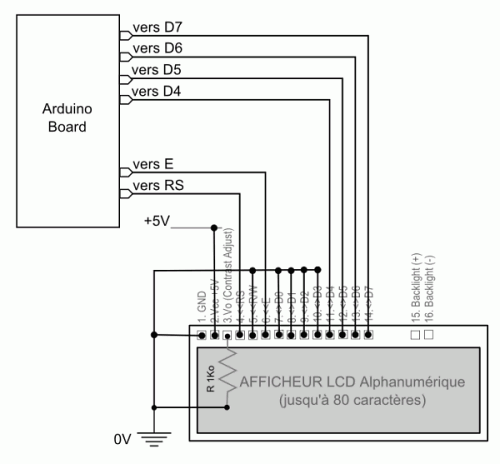
* un afficheur LCD alpha-numérique ([préparé pour une utilisation simplifiée en mode 4 bits](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.TechniquePreparationLCD) avec la carte Arduino)



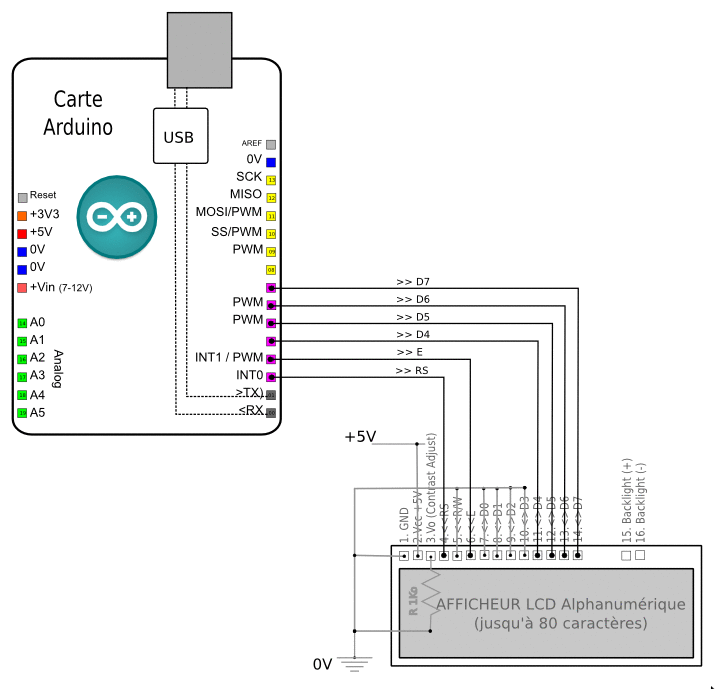
## 3.  Instructions de montage

* Connecter sur la broche 2 la broche RS du LCD
* Connecter sur la broche 3 la broche E du LCD
* Connecter sur la broche 4 la broche D4 du LCD
* Connecter sur la broche 5 la broche D5 du LCD
* Connecter sur la broche 6 la broche D6 du LCD
* Connecter sur la broche 7 la broche D7 du LCD

## 4.  Le schéma théorique du montage

  
Le schéma théorique du montage

## 5.  Le circuit du montage

[](http://www.mon-club-elec.fr/mes_images/montages/lcd/lcd_seul_pin2_7.gif)

Le schéma du montage à réaliser (cliquer pour agrandir)

## 6.  Au niveau de la partie déclarative :

#### Inclusion des librairies utilisées

* On inclut les librairies des fonctionnalités utilisées :
  + Inclusion de la librairie pour l'afficheur LCD alphanumérique :

#include <LiquidCrystal.h> // Inclusion de la librairie pour afficheur LCD

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=1)

#### Déclaration de constantes utiles

* On déclare les constantes utiles dans le programme :

#### Déclaration des constantes de broches

* Déclaration des constantes pour les broches utilisées dans le programme : les broches de commandes RS et E, les broches de données D4 à D7.

const int RS=2; //declaration constante de broche   
const int E=3; //declaration constante de broche   
const int D4=4; //declaration constante de broche   
const int D5=5; //declaration constante de broche   
const int D6=6; //declaration constante de broche   
const int D7=7; //declaration constante de broche

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=2)

#### Déclarations des objets utiles pour les fonctionnalités utilisées

* Déclaration des objets utiles pour les fonctionnalités utilisées :
  + Déclaration d'un objet LCD alphanumérique :

LiquidCrystal lcd(RS, E, D4, D5, D6, D7);// Création d'un objet LiquidCrystal = initialisation LCD en mode 4 bits

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=3)

## 7.  Au niveau de la fonction d'initialisation setup( ) :

#### Initialisation des fonctionnalités utilisées :

* On initialise les différentes fonctionnalités utilisées :
  + Initialisation de l'utilisation de l'afficheur LCD alpha-numérique

lcd.begin(20,4); // Initialise le LCD avec 20 colonnes x 4 lignes   
  
delay(10); // pause rapide pour laisser temps initialisation

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=4)

* Test de l'afficheur LCD alpha-numérique

// Test du LCD  
  
lcd.print("LCD OK") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.clear(); // // efface écran et met le curseur en haut à gauche  
delay(10); // pour laisser temps effacer écran

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=5)

#### configuration des broches utilisées :

* Configuration des broches en sortie : les broches utilisées avec le LCD sont configurées lors de l'initialisation du LCD ci-dessus.

## 8.  Au niveau de la boucle principale, la fonction loop ( ) :

* On affiche différents messages de texte et l'on teste différentes localisation sur le LCD :

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

[[Obtenir le Code]](http://www.mon-club-elec.fr/pmwiki_mon_club_elec/pmwiki.php?n=MAIN.ArduinoExpertLCDSeulAfficheTexte?action=sourceblock&num=6)

* Le programme recommence en boucle les instructions de la fonction loop( ).

Se reporter aux commentaires présents dans le programme pour plus de détails.

## 9.  Mise en oeuvre du programme

### 9.1  Préparation du montage et programmation de la carte Arduino :

* Commencer par réaliser le montage indiqué sur plaque d'expérimentation
* Ensuite, programmer la carte Arduino avec ce programme (en bas de page) selon la procédure habituelle

### 9.2  Fonctionnement

* Les messages de texte s'affiche sur le LCD.

## 10.  Le programme complet en langage Arduino :

A copier/coller directement dans l'éditeur Arduino

// --- Programme Arduino ---  
// par X. HINAULT - 01/2010   
  
// --- Que fait ce programme ? ---  
/\* Affiche des messages texte sur l'afficheur LCD\*/  
  
// --- Fonctionnalités utilisées ---  
// Utilise un afficheur LCD 4x20 en mode 4 bits   
  
// --- Circuit à réaliser ---  
  
// Connexion du LCD sur les broches de la carte Arduino   
// Connecter broche RS du LCD sur la broche 2  
// Connecter broche E du LCD sur la broche 3  
// Connecter broche D4 du LCD sur la broche 4  
// Connecter broche D5 du LCD sur la broche 5  
// Connecter broche D6 du LCD sur la broche 6  
// Connecter broche D7 du LCD sur la broche 7  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Entête déclarative \*\*\*\*\*\*\*  
// A ce niveau sont déclarées les librairies, les constantes, les variables...  
  
// --- Inclusion des librairies utilisées ---  
#include <LiquidCrystal.h> // Inclusion de la librairie pour afficheur LCD   
  
// --- Déclaration des constantes ---  
  
// --- constantes des broches ---  
  
const int RS=2; //declaration constante de broche   
const int E=3; //declaration constante de broche   
  
const int D4=4; //declaration constante de broche   
const int D5=5; //declaration constante de broche   
const int D6=6; //declaration constante de broche   
const int D7=7; //declaration constante de broche   
  
// --- Déclaration des variables globales ---  
  
// --- Initialisation des fonctionnalités utilisées ---  
LiquidCrystal lcd(RS, E, D4, D5, D6, D7);// initialisation LCD en mode 4 bits   
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FONCTION SETUP = Code d'initialisation \*\*\*\*\*  
// La fonction setup() est exécutée en premier et 1 seule fois, au démarrage du programme  
  
void setup()   { // debut de la fonction setup()  
  
// --- ici instructions à exécuter au démarrage ---   
  
lcd.begin(20,4); // Initialise le LCD avec 20 colonnes x 4 lignes   
  
delay(10); // pause rapide pour laisser temps initialisation  
  
// Test du LCD  
  
lcd.print("LCD OK") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.clear(); // // efface écran et met le curseur en haut à gauche  
delay(10); // pour laisser temps effacer écran  
  
} // fin de la fonction setup()  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* FONCTION LOOP = Boucle sans fin = coeur du programme \*\*\*\*\*\*\*\*\*\*\*\*\*  
// la fonction loop() s'exécute sans fin en boucle aussi longtemps que l'Arduino est sous tension  
  
void loop(){ // debut de la fonction loop()  
  
lcd.print("Arduino...") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.setCursor(9, 1) ; // 10ème col - 2ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)   
lcd.print("...pour te") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.setCursor(4, 2) ; // 5ème col - 3ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)   
lcd.print("...servir") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.setCursor(12, 3) ; // 13ème col - 4ème ligne - positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)   
lcd.print("...amigo") ; // affiche la chaîne texte - message de test  
delay(2000); // pause de 2 secondes  
  
lcd.clear(); // // efface écran et met le curseur en haut à gauche  
delay(10); // pour laisser temps effacer écran  
  
// --- ici instructions à exécuter par le programme principal ---   
  
} // fin de la fonction loop() - le programme recommence au début de la fonction loop sans fin  
// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
// --- Fin programme ---  
  
// --- Mémo instructions ---  
  
  
// ----- memo LCD ---   
// LiquidCrystal(rs, enable, d4, d5, d6, d7) ; // initialisation 4 bits  
// lcd.begin(cols, rows); // initialisation nombre colonne/ligne  
//   
// lcd.clear(); // efface écran et met le curseur en haut à gauche  
// lcd.home(); // repositionne le curseur en haut et à gauche SANS effacer écran  
//   
// lcd.setCursor(col, row) ; // positionne le curseur à l'endroit voulu (colonne, ligne) (1ère=0 !)   
// lcd.print("texte") ; // affiche la chaîne texte  
//   
// lcd.cursor() ; // affiche la ligne de base du curseur   
// lcd.noCursor() ; // cache le curseur   
// lcd.blink() ; // fait clignoter le curseur  
// lcd.noBlink() ;// stoppe le clignotement du curseur  
// lcd.noDisplay() ; // éteint le LCD sans modifier affichage  
// lcd.display() ; // rallume le LCD sans modif affichage  
//   
// lcd.scrollDisplayLeft(); // décale l'affichage d'une colonne vers la gauche  
// lcd.scrollDisplayRight(); // décale l'affichage d'une colonne vers la droite   
// lcd.autoscroll() ; // les nouveaux caractères poussent les caractères déjà affichés  
// noAutoscroll(); // stoppe le mode autoscroll

# Annexes

# <http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/introduire-le-temps>

Introduire le temps

C'est bien beau d'allumer une LED, mais si elle ne fait rien d'autre, ce n'est pas très utile. Autant la brancher directement sur une pile (avec une résistance tout de même ! :p). Alors voyons comment rendre intéressante cette LED en la faisant clignoter ! Ce que ne sait pas faire une pile...

Pour cela il va nous falloir introduire la notion de temps. Et bien devinez quoi ? Il existe une fonction toute prête là encore ! Je ne vous en dis pas plus, passons à la pratique !

### Comment faire ?

##### Trouver la commande...

Je vous laisse cherche un peu par vous même, cela vous entrainera ! :pirate:

...

Pour ceux qui ont fait l'effort de chercher et n'ont pas trouvé (à cause de l'anglais ?), je vous donne la fonction qui va bien :

On va utiliser : **delay()**

Petite description de la fonction, elle va servir à mettre en pause le programme pendant un temps prédéterminé.

##### Utiliser la commande

La fonction admet un paramètre qui est le temps pendant lequel on veut mettre en pause le programme. Ce temps doit être donné en millisecondes. C'est-à-dire que si vous voulez arrêter le programme pendant 1 seconde, il va falloir donner à la fonction ce même temps, écrit en millisecondes, soit 1000ms.

Le code est simple à utiliser, il est le suivant :

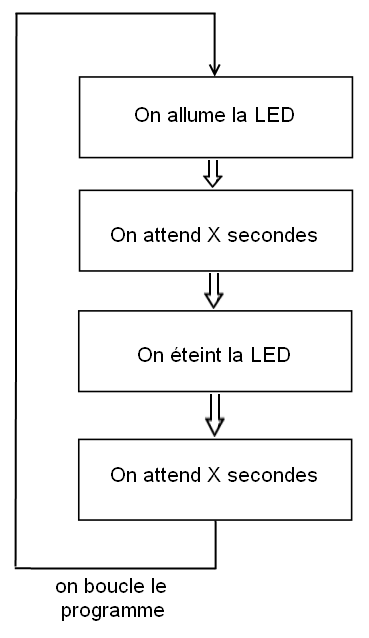
delay(1000);  // on fait une pause du programme pendant 1000ms, soit 1 seconde

Rien de plus simple donc. Pour 20 secondes de pause, il aurait fallu écrire :

delay(20000);  // on fait une pause du programme pendant 20000ms, soit 20 secondes

##### Mettre en pratique : faire clignoter une LED

Du coup, si on veut faire clignoter notre LED, il va falloir utiliser cette fonction. Voyons un peu le schéma de principe du clignotement d'une LED :



Vous le voyez, la LED s'allume. Puis, on fait intervenir la fonction **delay()**, qui va mettre le programme en pause pendant un certain temps. Ensuite, on éteint la LED. On met en pause le programme. Puis on revient au début du programme. On recommence et ainsi de suite. C'est cette somme de commande, qui forme le processus qui fait clignoter la LED.

Dorénavant, prenez l'habitude de faire ce genre de schéma lorsque vous faites un programme. Cela aide grandement la réflexion, croyez moi ! ;)C'est le principe de perdre du temps pour en gagner. Autrement dit : l'**organisation** !

Maintenant, il faut que l'on traduise ce schéma, portant le nom d'**organigramme**, en code. Il suffit pour cela de remplacer les phrases dans chaque cadre par une ligne de code.

Par exemple, "on allume la LED", va être traduis par l'instruction que l'on a vue dans le chapitre précédent :

digitalWrite(led\_rouge, LOW);   // allume la LED

Ensuite, on traduit le cadre suivant, ce qui donne :

delay(1000);   // fait une pause de 1 seconde (= 1000ms)

Puis, on traduit la ligne suivante :

digitalWrite(led\_rouge, HIGH);   // éteint la LED

Enfin, la dernière ligne est identique à la deuxième, soit :

delay(1000);   // fait une pause de 1 seconde

On se retrouve avec le code suivant :

digitalWrite(led\_rouge, LOW);   // allume la LED

delay(1000);   // fait une pause de 1 seconde

digitalWrite(led\_rouge, HIGH);   // éteint la LED

delay(1000);   // fait une pause de 1 seconde

La fonction qui va boucler à l'infini le code précédent est la fonction **loop()**. On inscrit donc le code précédent dans cette fonction :

void loop()

{

   digitalWrite(led\_rouge, LOW);   // allume la LED

   delay(1000);   // fait une pause de 1 seconde

   digitalWrite(led\_rouge, HIGH);   // éteint la LED

   delay(1000);   // fait une pause de 1 seconde

}

Et on n'oublie pas de définir la broche utilisée par la LED, ainsi que d'initialiser cette broche en tant que sortie. Cette fois, le code est terminé !

const int led\_rouge = 2;       //définition de la broche 2 de la carte en tant que variable

void setup()                   //fonction d'initialisation de la carte

{

   pinMode(led\_rouge, OUTPUT); //initialisation de la broche 2 comme étant une sortie

}

void loop()                    //fonction principale, elle se répète (s’exécute) à l'infini

{

   digitalWrite(led\_rouge, LOW);    // allume la LED

   delay(1000);                      // fait une pause de 1 seconde

   digitalWrite(led\_rouge, HIGH);     // éteint la LED

   delay(1000);                      // fait une pause de 1 seconde

}

Vous n'avez plus qu'à charger le code dans la carte et admirer mon votre travail ! La LED clignote ! Libre à vous de changer le temps de clignotement : vous pouvez par exemple éteindre la LED pendant 40ms et l'allumer pendant 600ms :

const int led\_rouge = 2;       //définition de la broche 2 de la carte en tant que variable

void setup()                   //fonction d'initialisation de la carte

{

   pinMode(led\_rouge, OUTPUT); //initialisation de la broche 2 comme étant une sortie

}

void loop()                    //fonction principale, elle se répète (s’exécute) à l'infini

{

   digitalWrite(led\_rouge, LOW);    // allume la LED

   delay(600);                      // fait une pause de 600 milli-seconde

   digitalWrite(led\_rouge, HIGH);     // éteint la LED

   delay(40);                      // fait une pause de 40 milli-seconde

}

Et Hop, une petite vidéo d'illustration !  
<http://www.youtube.com/v/YAOakcEoIfk>

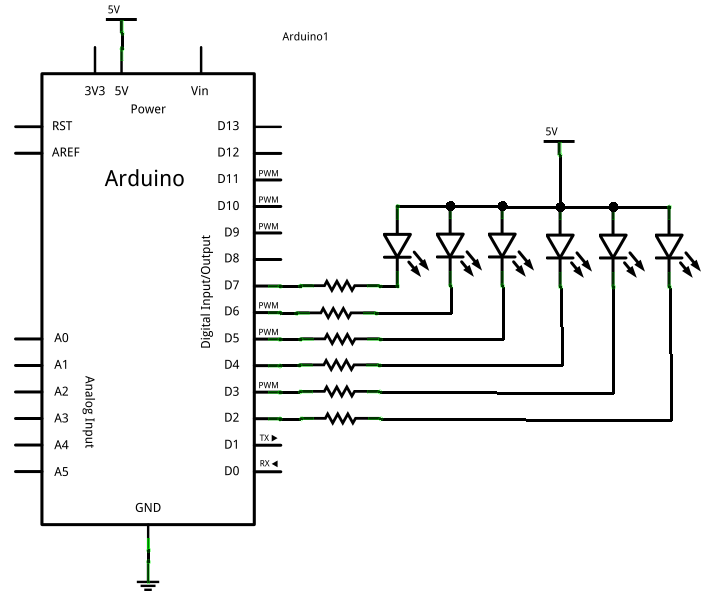
### Faire clignoter un groupe de LED

Vous avouerez facilement que ce n'était pas bien difficile d'arriver jusque-là. Alors, à présent, accentuons la difficulté. Notre but : faire clignoter un groupe de LED.

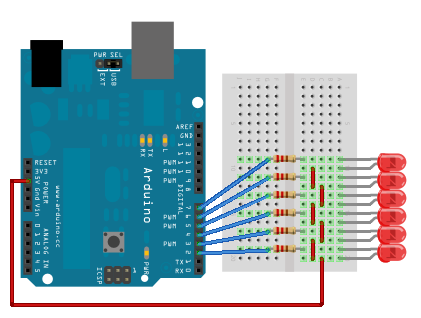
##### Le matériel et les schémas

Ce groupe de LED sera composé de six LED, nommées L1, L2, L3, L4, L5 et L6. Vous aurez par conséquent besoin d'un nombre semblable de résistances.

Le schéma de la réalisation :

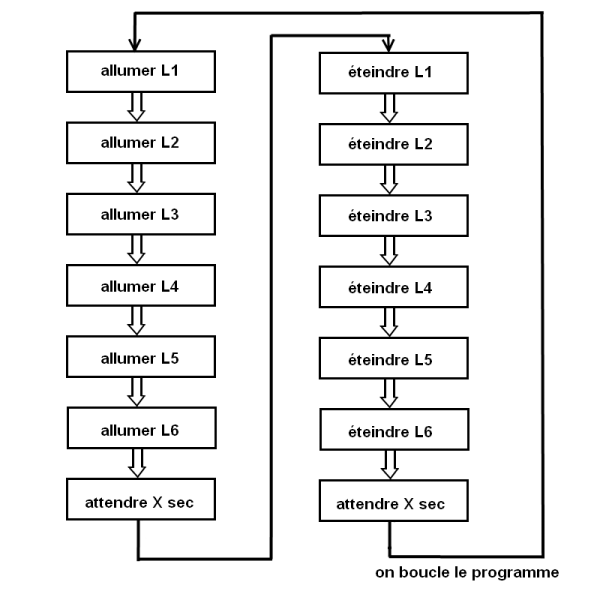


La photo de la réalisation :



##### Le programme

Le programme est un peu plus long que le précédent, car il ne s'agit plus d'allumer 1 seule LED, mais 6 ! Voilà l'organigramme que va suivre notre programme :



Cet organigramme n'est pas très beau, mais il a le mérite d'être assez lisible. Nous allons essayer de le suivre pour créer notre programme.

Traduction des six premières instructions :

digitalWrite(L1, LOW);  //notez que le nom de la broche à changé

digitalWrite(L2, LOW);  //et ce pour toutes les LED connectées

digitalWrite(L3, LOW);  //au micro-controleur

digitalWrite(L4, LOW);

digitalWrite(L5, LOW);

digitalWrite(L6, LOW);

Ensuite, on attend 1,5 seconde :

delay(1500);

Puis on traduis les six autres instructions :

digitalWrite(L1, HIGH);  //on éteint les LED

digitalWrite(L2, HIGH);

digitalWrite(L3, HIGH);

digitalWrite(L4, HIGH);

digitalWrite(L5, HIGH);

digitalWrite(L6, HIGH);

Enfin, la dernière ligne de code, disons que nous attendrons 4,32 secondes :

delay(4320);

Tous ces bouts de code sont à mettre à la suite et dans la fonction **loop()** pour qu'ils se répètent.

void loop()

{

   digitalWrite(L1, LOW);  //allumer les LED

   digitalWrite(L2, LOW);

   digitalWrite(L3, LOW);

   digitalWrite(L4, LOW);

   digitalWrite(L5, LOW);

   digitalWrite(L6, LOW);

   delay(1500);             //attente du programme de 1,5 secondes

   digitalWrite(L1, HIGH);   //on éteint les LED

   digitalWrite(L2, HIGH);

   digitalWrite(L3, HIGH);

   digitalWrite(L4, HIGH);

   digitalWrite(L5, HIGH);

   digitalWrite(L6, HIGH);

   delay(4320);             //attente du programme de 4,32 secondes

}

Je l'ai mentionné dans un de mes commentaires entre les lignes du programme, les noms attribués aux broches sont à changer. En effet, car si on définit des noms de variables identiques, le compilateur n'aimera pas ça et vous affichera une erreur. En plus, le micro-contrôleur ne pourrait pas exécuter le programme car il ne saurait pas quelle broche mettre à l'état HAUT ou BAS.

Pour définir les broches, on fait la même chose qu'à notre premier programme :

const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1

const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2

const int L3 = 4; // ...

const int L4 = 5;

const int L5 = 6;

const int L6 = 7;

Maintenant que les broches utilisées sont définies, il faut dire si ce sont des entrées ou des sorties :

pinMode(L1, OUTPUT); //L1 est une broche de sortie

pinMode(L2, OUTPUT); //L2 est une broche de sortie

pinMode(L3, OUTPUT); // ...

pinMode(L4, OUTPUT);

pinMode(L5, OUTPUT);

pinMode(L6, OUTPUT);

##### Le programme final

Il n'est certes pas très beau, mais il fonctionne :

const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1

const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2

const int L3 = 4; // ...

const int L4 = 5;

const int L5 = 6;

const int L6 = 7;

void setup()

{

   pinMode(L1, OUTPUT); //L1 est une broche de sortie

   pinMode(L2, OUTPUT); //L2 est une broche de sortie

   pinMode(L3, OUTPUT); // ...

   pinMode(L4, OUTPUT);

   pinMode(L5, OUTPUT);

   pinMode(L6, OUTPUT);

}

void loop()

{

   digitalWrite(L1, LOW);  //allumer les LED

   digitalWrite(L2, LOW);

   digitalWrite(L3, LOW);

   digitalWrite(L4, LOW);

   digitalWrite(L5, LOW);

   digitalWrite(L6, LOW);

   delay(1500);             //attente du programme de 1,5 secondes

   digitalWrite(L1, HIGH);   //on éteint les LED

   digitalWrite(L2, HIGH);

   digitalWrite(L3, HIGH);

   digitalWrite(L4, HIGH);

   digitalWrite(L5, HIGH);

   digitalWrite(L6, HIGH);

   delay(4320);             //attente du programme de 4,32 secondes

}

Voilà, vous avez en votre possession un magnifique clignotant, que vous pouvez attacher à votre vélo ! :p

Une question me chiffonne. Doit-on toujours écrire l'état d'une sortie, ou peut-on faire plus simple ?

Tu soulèves un point intéressant. Si je comprends bien, tu te demandes comment faire pour remplacer l’intérieur de la fonction **loop()**? C'est vrai que c'est très lourd à écrire et à lire ! Il faut en effet s'occuper de définir l'état de chaque LED. C'est rébarbatif, surtout si vous en aviez mis autant qu'il y a de broches disponibles sur la carte !

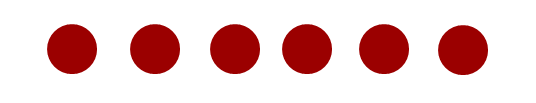
Il y a une solution pour faire ce que tu dis. Nous allons la voir dans quelques chapitres, ne sois pas impatient ! ;)

En attendant, voici une vidéo d'illustration du clignotement :  
<http://www.youtube.com/v/2SgxKU67mn8>

### Réaliser un chenillard

##### Le but du programme

Le but du programme que nous allons créer va consister à réaliser un chenillard. Pour ceux qui ne savent pas ce qu'est un chenillard, je vous ai préparé une petite image .gif animée :



Comme on dit souvent, une image vaut mieux qu'un long discours ! :p

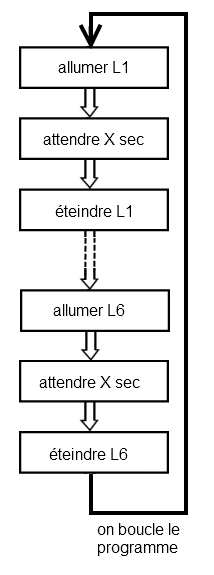
Voilà donc ce qu'est un chenillard. Chaque LED s'allume alternativement et dans l'ordre chronologique. De la gauche vers la droite ou l'inverse, c'est au choix.

##### Organigramme

Comme j'en ai marre de faire des dessins avec paint.net, je vous laisse réfléchir tout seul comme des grands à l'organigramme du programme.

...

Bon, aller, le voilà cet organigramme ! Attention, il n'est pas complet, mais si vous avez compris le principe, le compléter ne vous posera pas de problèmes :



A vous de jouer !

##### Le programme

Normalement, sa conception ne devrait pas vous poser de problèmes. Il suffit en effet de récupérer le code du programme précédent ("allumer un groupe de LED") et de le modifier en fonction de notre besoin.

Ce code, je vous le donne, avec les commentaires qui vont bien :

// on garde le même début que le programme précédent

const int L1 = 2; //broche 2 du micro-contrôleur se nomme maintenant : L1

const int L2 = 3; //broche 3 du micro-contrôleur se nomme maintenant : L2

const int L3 = 4; // ...

const int L4 = 5;

const int L5 = 6;

const int L6 = 7;

void setup()

{

   pinMode(L1, OUTPUT); //L1 est une broche de sortie

   pinMode(L2, OUTPUT); //L2 est une broche de sortie

   pinMode(L3, OUTPUT); // ...

   pinMode(L4, OUTPUT);

   pinMode(L5, OUTPUT);

   pinMode(L6, OUTPUT);

}

// on change simplement l’intérieur de la boucle pour atteindre notre objectif

void loop() //la fonction loop() exécute le code qui suit en le répétant en boucle

{

   digitalWrite(L1, LOW);   //allumer L1

   delay(1000);             //attendre 1 seconde

   digitalWrite(L1, HIGH);  //on éteint L1

   digitalWrite(L2, LOW);   //on allume L2 en même temps que l'on éteint L1

   delay(1000);             //on attend 1 seconde

   digitalWrite(L2, HIGH);  //on éteint L2 et

   digitalWrite(L3, LOW);   //on allume immédiatement L3

   delay(1000);             // ...

   digitalWrite(L3, HIGH);

   digitalWrite(L4, LOW);

   delay(1000);

   digitalWrite(L4, HIGH);

   digitalWrite(L5, LOW);

   delay(1000);

   digitalWrite(L5, HIGH);

   digitalWrite(L6, LOW);

   delay(1000);

   digitalWrite(L6, HIGH);

}

Vous le voyez, ce code est très lourd et n'est pas pratique. Nous verrons plus loin comment faire en sorte de l’alléger. Mais avant cela, un TP arrive...

Au fait, voici un exemple de ce que vous pouvez obtenir !  
<http://www.youtube.com/v/hKNqRAi-kKI>

### Fonction millis()

Nous allons terminer ce chapitre par un point qui peutêtre utile, notamment dans certaines situations où l'on veut ne pas arrêter le programme. En effet, si on veut faire clignoter une LED sans arrêter l’exécution du programme, on ne peut pas utiliser la fonction delay() qui met en pause le programme durant le temps défini.

#### Les limites de la fonction delay()

Vous avez probablement remarqué, lorsque vous utilisez la fonction "delay()" tout notre programme s’arrête le temps d'attendre. Dans certains cas ce n'est pas un problème mais dans certains cas ça peut être plus gênant.

Imaginons, vous êtes en train de faire avancer un robot. Vous mettez vos moteurs à une vitesse moyenne, tranquille, jusqu'à ce qu'un petit bouton sur l'avant soit appuyé (il clic lorsqu'on touche un mur par exemple). Pendant ce temps-là, vous décidez de faire des signaux en faisant clignoter vos LED. Pour faire un joli clignotement, vous allumez une LED rouge pendant une seconde puis l’éteignez pendant une autre seconde. Voilà par exemple ce qu'on pourrait faire comme code

void setup()

{

    pinMode(moteur, OUTPUT);

    pinMode(led, OUTPUT);

    pinMode(bouton, INPUT);

    digitalWrite(moteur, HIGH); //on met le moteur en marche (en admettant qu'il soit en marche à HIGH)

    digitalWrite(led, LOW); //on allume la LED

}

void loop()

{

    if(digitalRead(bouton)==HIGH) //si le bouton est cliqué (on rentre dans un mur)

    {

        digitalWrite(moteur, LOW); //on arrête le moteur

    }

    else //sinon on clignote

    {

        digitalWrite(led, HIGH);

        delay(1000);

        digitalWrite(led, LOW);

        delay(1000);

    }

}

Attention ce code n'est pas du tout rigoureux voire faux dans son écriture, il sert juste à comprendre le principe !

Maintenant imaginez. Vous roulez, tester que le bouton n'est pas appuyé, donc faites clignoter les LED (cas du else). Le temps que vous fassiez l'affichage en entier s'écoule 2 longues secondes ! Le robot a pu pendant cette éternité se prendre le mur en pleine poire et les moteurs continuent à avancer tête baissée jusqu'à fumer ! Ce n'est pas bon du tout !

Voici pourquoi la fonction millis() peut nous sauver.

#### Découvrons et utilisons millis()

Tout d'abord, quelques précisions à son sujet, avant d'aller s'en servir. A l'intérieur du cœur de la carte Arduino se trouve un chronomètre. Ce chrono mesure l'écoulement du temps depuis le lancement de l'application. Sa granularité (la précision de son temps) est la milliseconde. La fonction millis() nous sert à savoir quelle est la valeur courante de ce compteur. Attention, comme ce compteur est capable de mesurer une durée allant jusqu'à 50 jours, la valeur retournée doit être stockée dans une variable de type "long".

C'est bien gentil mais concrètement on l'utilise comment ?

Et bien c'est très simple. On sait maintenant "lire l'heure". Maintenant, au lieu de dire "allume-toi pendant une seconde et ne fais surtout rien pendant ce temps", on va faire un truc du genre "Allume-toi, fais tes petites affaires, vérifie l'heure de temps en temps et si une seconde est écoulée, alors réagis !".

Voici le code précédent transformé selon la nouvelle philosophie :

long temps; //variable qui stocke la mesure du temps

boolean etat\_led;

void setup()

{

    pinMode(moteur, OUTPUT);

    pinMode(led, OUTPUT);

    pinMode(bouton, INPUT);

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

    etat\_led = 0; // par défaut la LED sera éteinte

    digitalWrite(led, etat\_led); //on éteint la LED

}

void loop()

{

    if(digitalRead(bouton)==HIGH) //si le bouton est cliqué (on rentre dans un mur)

    {

        digitalWrite(moteur, LOW); //on arrête le moteur

    }

    else //sinon on clignote

    {

        //on compare l'ancienne valeur du temps et la valeur sauvée

        //si la comparaison (l'un moins l'autre) dépasse 1000...

        //...cela signifie qu'au moins une seconde s'est écoulée

        if((millis() - temps) > 1000)

        {

            etat\_led = !etat\_led; //on inverse l'état de la LED

            digitalWrite(led, etat\_led); //on allume ou éteint

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

        }

    }

}

Et voilà, grâce à cette astuce plus de fonction bloquante. L'état du bouton est vérifié très fréquemment ce qui permet de s'assurer que si jamais on rentre dans un mur, on coupe les moteurs très vite. Dans ce code, tout s'effectue de manière fréquente. En effet, on ne reste jamais bloqué à attendre que le temps passe. A la place, on avance dans le programme et test souvent la valeur du chronomètre. Si cette valeur est de 1000 itérations supérieures à la dernière valeur mesurée, alors cela signifie qu'une seconde est passée.

Attention, au "if" de la ligne 25 ne faites surtout pas "millis() - temp == 1000". Cela signifierait que vous voulez vérifier que 1000 millisecondes EXACTEMENT se sont écoulées, ce qui est très peu probable (vous pourrez plus probablement mesurer plus ou moins mais rarement exactement)

# http://fr.openclassrooms.com/sciences/cours/arduino-pour-bien-commencer-en-electronique-et-en-programmation/ajouter-des-sorties-numeriques-a-l-arduino

## Ajouter des sorties (numériques) à l'Arduino

Dans ce chapitre "bonus", nous allons vous faire découvrir comment ajouter des sorties numériques à votre carte Arduino. Car en effet, pour vos projets les plus fous, vous serez certainement amené à avoir besoin d'un grand nombre de sorties. Là il y a deux choix : le premier serait d'opter pour une carte Arduino qui dispose de plus de sorties, telle que la Arduino Mega ; mais dans le cas où vous aurez besoin d'un giga super ultra grand nombre de sorties, même la Mega ne suffira pas. Le deuxième choix c'est donc... de lire ce chapitre. :-°

Ce que vous allez découvrir se révélera fort utile, soyez-en certains. Prenons l'exemple suivant : dans le cas où vous devrez gérer un grand nombre de LED pour réaliser un afficheur comme l'on en trouve parfois dans les vitrines de magasins, vous serez très vite limité par le nombre de sorties de votre Arduino. Surtout si votre afficheur contient plus de 1000 LED ! Ce chapitre va alors vous aider dans de pareils cas, car nous allons vous présenter un composant spécialisé dans ce domaine : le **74HC595**.

### Présentation du 74HC595

#### Principe

Comme je viens de l’énoncer, il peut arriver qu'il vous faille utiliser plus de broches qu'il n'en existe sur un micro-contrôleur, votre carte Arduino en l'occurrence (ou plutôt, l'ATMEGA328 présent sur votre carte Arduino). Dans cette idée, des ingénieurs ont développé un composant que l'on pourrait qualifier de "décodeur série -> parallèle". D'une manière assez simple, cela consiste à envoyer un octet de données (8 bits) à ce composant qui va alors décoder l'information reçue et changer l'état de chacune de ses sorties en conséquence. Le composant que nous avons choisi de vous faire utiliser dispose de huit sorties de données pour une seule entrée de données.

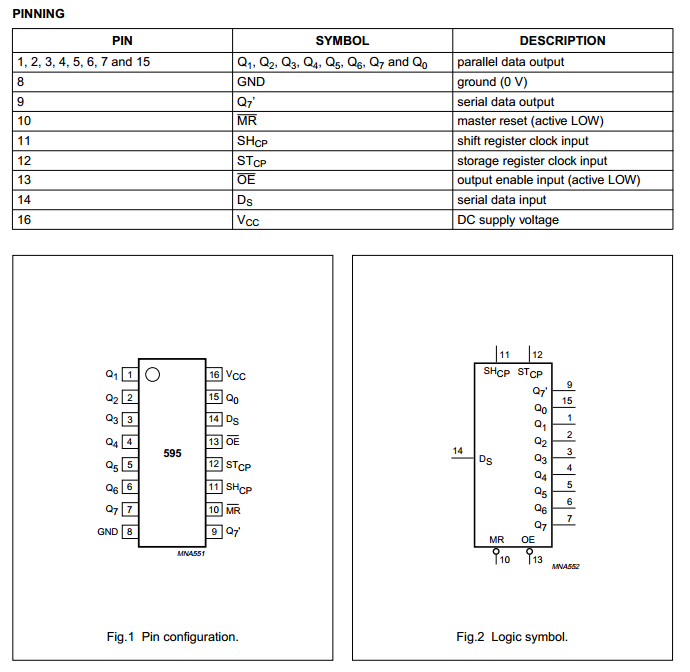
Concrètement, cela signifie que lorsque l'on enverra l'octet suivant : 00011000 au décodeur 74HC595, il va changer l'état (HAUT ou BAS) de ses sorties. On verra alors, en supposant qu'il y a une LED de connectée sur chacune de ses sorties, les 2 LED du "milieu" (géographiquement parlant) qui seront dans un état opposé de leurs congénères. Ainsi, en utilisant seulement deux sorties de votre carte Arduino, on peut virtuellement en utiliser 8 (voir beaucoup plus mais nous verrons cela plus tard).

#### Le composant

Rentrons maintenant dans les entrailles de ce fameux 595. Pour cela nous utiliserons [cette datasheet](http://www.datasheetcatalog.org/datasheet/philips/74AHC_AHCT595_1.pdf) tout au long du tuto.

##### Brochage

Lisons ensemble quelques pages.  
La première nous donne, de par le titre, la fonctionnalité du composant. Elle est importante car l'on sait à ce moment à quel composant nous allons avoir affaire.  
La seconde apporte déjà quelques informations utiles outre la fonctionnalité. Au-delà du résumé qu'il est toujours bon de lire, les caractéristiques du composant sont détaillées. On apprend également que ce composant peut fonctionner jusqu'à une fréquence de 170MHz. C'est très très rapide par rapport à notre carte Arduino qui tourne à 16MHz, nous sommes tranquilles de ce côté-là.  
Continuons...  
C'est la page 4 qui nous intéresse vraiment ici. On y retrouve le tableau et la figure suivante :

Brochage du 595

Avec ce dernier, on va pouvoir faire le lien entre le nom de chaque broche et leur rôle. De plus, nous savons où elles sont placées sur le composant. Nous avons donc les sorties et la masse à gauche et les broches de commande à droite (plus la sortie Q0) et l'alimentation.  
Voyons maintenant comment faire fonctionner tout cela.

##### Fonctionnement

Comme tout composant électronique, il faut commencer par l'alimenter pour le faire fonctionner. Le tableau que nous avons vu juste au-dessus nous indique que les broches d'alimentation sont la broche 16 (VCC) et la broche 8 (masse). Quelques pages plus loin dans la datasheet, page 7 précisément, nous voyons la tension à appliquer pour l'alimenter : entre 2V et 5.5V (et idéalement 5.0V). Une fois que ce dernier est alimenté, il faut se renseigner sur le rôle des broches pour savoir comment l'utiliser correctement. Pour cela il faut revenir sur le tableau précédent et la table de vérité qui le suit.

On découvre donc que les sorties sont les broches de 1 à 7 et la broche 15 (Qn) ; l'entrée des données série, qui va commander les sorties du composant, se trouve sur la broche 14 (serial data input) ; une sortie particulière est disponible sur la broche 9 (serial data output, nous y reviendrons à la fin de ce chapitre).

Sur la broche 10 on trouve le Master Reset, pour mettre à zéro toutes les sorties. Elle est active à l'état BAS. Vous ferez alors attention, dans le cas où vous utiliseriez cette sortie, de la forcer à un état logique HAUT, en la reliant par exemple au +5V ou bien à une broche de l'Arduino que vous ne mettrez à l'état BAS que lorsque vous voudrez mettre toutes les sorties du 74HC595 à l'état bas. Nous, nous mettrons cette sortie sur le +5V.

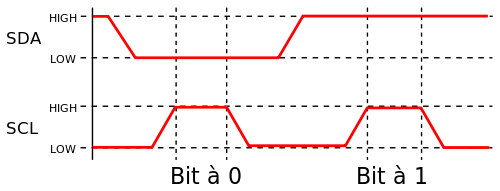
La broche 13, output enable input, est un broche de sélection qui permet d'inhiber les sorties. En clair, cela signifie que lorsque cette broche n'a pas l'état logique requis, les sorties du 74HC595 ne seront pas utilisables. Soit vous choisissez de l'utiliser en la connectant à une sortie de l'Arduino, soit on la force à l'état logique BAS pour utiliser pleinement chaque sortie. Nous, nous la relierons à la masse.

Deux dernières broches sont importantes. La n°11 et la n°12. Ce sont des "horloges". Nous allons expliquer quelle fonction elles remplissent.

Lorsque nous envoyons un ordre au 74HC595, nous envoyons cet ordre sous forme d'états logiques qui se suivent. Par exemple l'ordre 01100011. Cet ordre est composé de 8 états logiques, ou bits, et forme un octet. Cet ordre va précisément définir l'état de sortie de chacune des sorties du 74HC595. Le problème c'est que ce composant ne peut pas dissocier chaque bit qui arrive.

Prenons le cas des trois zéros qui se suivent dans l'octet que nous envoyons. On envoie le premier 0, la tension sur la ligne est alors de 0V. Le second 0 est envoyé, la tension est toujours de 0V. Enfin le dernier zéro est envoyé, avec la même tension de 0V puis vient un changement de tension à 5V avec l'envoi du 1 qui suit les trois 0. Au final, le composant n'aura vu en entrée qu'un seul 0 puisqu'il n'y a eu aucun changement d'état. De plus, il ne peut pas savoir quelle est la durée des états logiques qu'on lui envoie. S'il le connaissait, ce temps de "vie" des états logiques qu'on lui envoie, il pourrait aisément décoder l'ordre transmis. En effet, il pourrait se dire: "tiens ce bit (état logique) dépasse 10ms, donc un deuxième bit l'accompagne et est aussi au niveau logique 0". Encore 10ms d'écoulée et toujours pas de changement, eh bien c'est un troisième bit au niveau 0 qui vient d'arriver. C'est dans ce cas de figure que l'ordre reçu sera compris dans sa totalité par le composant.

Bon, eh bien c'est là qu'intervient le **signal d'horloge**. Ce signal est en fait là dans l'unique but de dire si c'est un nouveau bit qui arrive, puisque le 74HC595 n'est pas capable de le voir tout seul. En fait, c'est très simple, l'horloge est un signal carré fixé à une certaine fréquence. À chaque front montant (quand le signal d'horloge passe du niveau 0 au niveau 1), le 74HC595 saura que sur son entrée, c'est un nouveau bit qui arrive. Il pourra alors facilement voir s'il y a trois 0 qui se suivent. Ce chronogramme vous aidera à mettre du concret dans vos idées :



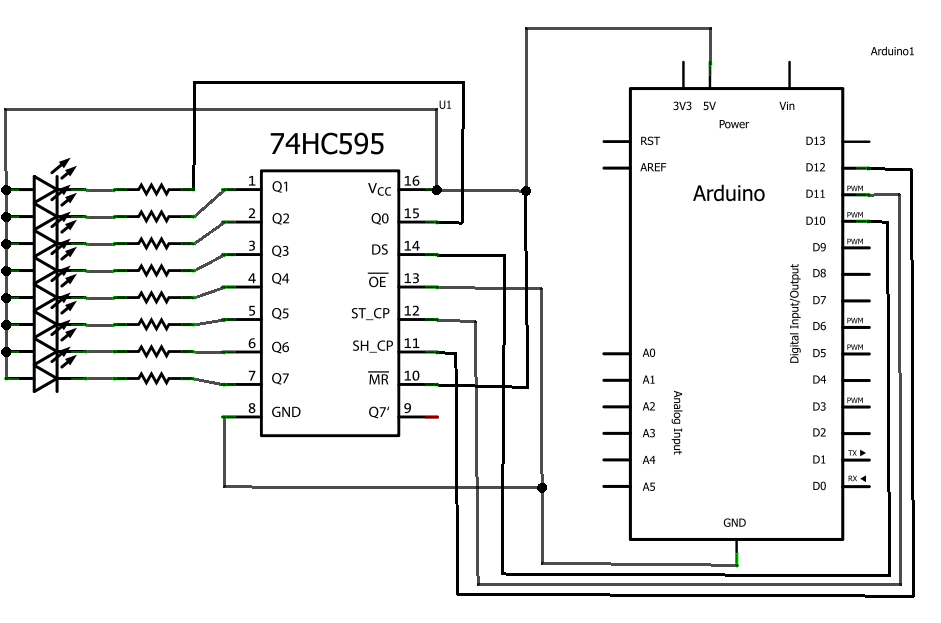
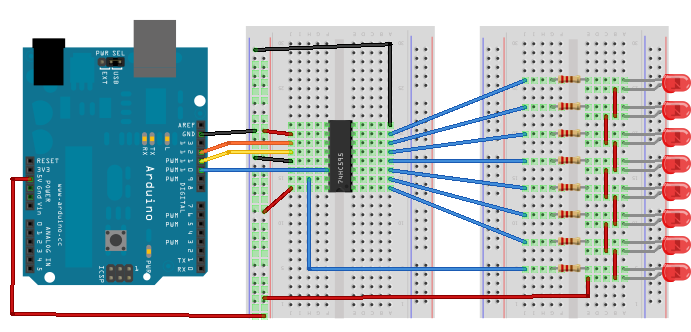
Source : Wikipédia -  
SDA est le signal de données, l'ordre que l'on envoie ; SCL est le signal d'horloge

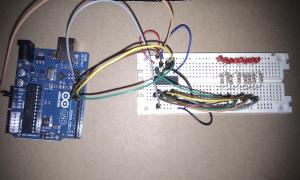
Pour câbler cette horloge, il faudra connecter une broche de l'Arduino à la broche numéro 11 du 74HC595. Ce signal travaillera donc en corrélation avec le signal de données relié sur la broche 14 du composant.

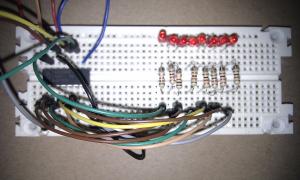
La seconde horloge pourrait aussi s'appeler "verrou". Elle sert à déterminer si le composant doit mettre à jour les états de ses sorties ou non, en fonction de l'ordre qui est transmis. Lorsque ce signal passe de l'état BAS à l'état HAUT, le composant change les niveaux logiques de ses sorties en fonction des bits de données reçues. En clair, il copie les huit derniers bits transmis sur ses sorties. Ce verrou se présente sur la broche 12.

##### Montage

Voici un petit montage à titre d'illustration que nous utiliserons par la suite. Je vous laisse faire le câblage sur votre breadboard comme bon vous semble, pendant ce temps je vais aller me siroter un bon petit café. :-°

Montage du 595 schémaMontage du 595 breadboard

[[](http://uploads.siteduzero.com/files/406001_407000/406175.jpg)Montage du HC595 et 8 LEDs](http://uploads.siteduzero.com/files/406001_407000/406175.jpg)

[[](http://uploads.siteduzero.com/files/406001_407000/406176.jpg)Montage du HC595 et 8 LEDs (zoom)](http://uploads.siteduzero.com/files/406001_407000/406176.jpg)

### Programmons pour utiliser ce composant

#### Envoyer un ordre au 74HC595

Nous allons maintenant voir comment utiliser le composant de manière logicielle, avec Arduino. Pour cela, je vais vous expliquer la façon de faire pour lui envoyer un ordre. Puis, nous créerons nous-mêmes la fonction qui va commander le 74HC595.

##### Le protocole

Nous le verrons dans le chapitre sur la liaison série plus en détail, le protocole est en fait un moyen qui permet de faire communiquer deux dispositifs. C'est une sorte de convention qui établit des règles de langage. Par exemple, si deux personnes parlent deux langues différentes, elles vont avoir un mal fou à se comprendre l'une de l'autre. Eh bien le protocole sert à imposer un langage qui leur permettra de se comprendre. En l'occurrence, il va s'agir de l'anglais.

Bon, cet exemple n'est pas parfait et a ses limites, c'est avant tout pour vous donner une vague idée de ce qu'est un protocole. Comme je vous l'ai dit, on en reparlera dans la partie suivante.

Nous l'avons vu tout à l'heure, pour envoyer un ordre au composant, il faut lui transmettre une série de bits. Autrement dit, il faut envoyer des bits les uns après les autres sur la même broche d'entrée. Cette broche sera nommée "data".

Ensuite, rappelez-vous, le composant a besoin de savoir quand lire la donnée, quand est-ce qu'un nouveau bit est arrivé ? C'est donc le rôle de l'horloge, ce que je vous expliquais plus haut. On pourrait s'imaginer qu'elle dit au composant : " Top ! tu peux lire la valeur car c'est un autre bit qui arrive sur ton entrée ! ".

Enfin, une troisième broche où l'on va amener l'horloge de verrou sert à dire au composant : " Nous sommes en train de mettre à jour la valeur de tes sorties, alors le temps de la mise à jour, garde chaque sortie à son état actuel ". Quand elle changera d'état, en passant du niveau BAS au niveau HAUT (front montant), cela donnera le "top" au composant pour qu'il puisse mettre à jour ses sorties avec les nouvelles valeurs.

Si jamais vous voulez économiser une broche sur votre Arduino, l'horloge de verrou peut être reliée avec l'horloge de données. Dans ce cas l'affichage va "scintiller" lors de la mise à jour car les sorties seront rafraîchies en même temps que la donnée arrive. Ce n'est pas gênant pour faire de l'affichage sur des LEDs mais ça peut l'être beaucoup plus si on a un composant qui réagit en fonction du 595.

##### Création de la fonction d'envoi

Passons à la création de la fonction d'envoi des données. C'est avec cette fonction que nous enverrons les ordres au 74HC595, pour lui dire par exemple d'allumer une LED sur sa sortie 4. On va donc faire un peu de programmation, aller zou !

Commençons par nommer judicieusement cette fonction : envoi\_ordre().

Cette fonction va prendre quatre paramètres. Le premier sera le numéro de la broche de données. Nous l’appellerons "dataPin". Le second sera similaire puisque ce sera le numéro de la broche d'horloge. Nous l'appellerons "clockPin". Le troisième sera le "sens" d'envoi des données, je reviendrai là-dessus ensuite. Enfin le dernier paramètre sera la donnée elle-même, donc un char (sur 8 bits, exactement comme l'ordre qui est à envoyer), que nous appellerons "donnee". Le prototype de la fonction sera alors le suivant :

void envoi\_ordre(int dataPin, int clockPin, boolean sens, char donnee)

Le code de la fonction ne sera pas très compliqué. Comme expliqué plus tôt, il suffit de générer une horloge et d'envoyer la bonne donnée pour que tout se passe bien.  
Le 74HC595 copie le bit envoyé dans sa mémoire lorsque le signal d'horloge passe de 0 à 1. Pour cela, il faut donc débuter le cycle par une horloge à 0. Ensuite, nous allons placer la donnée sur la broche de donnée. Enfin, nous ferons basculer la broche d'horloge à l'état haut pour terminer le cycle. Nous ferons ça huit fois pour pouvoir envoyer les huit bits de l'octet concerné (l'octet d'ordre). Schématiquement le code serait donc le suivant :

for(int i=0; i<8; i++) //on va parcourir chaque bit de l'octet

  {

     //départ du cycle, on met l'horloge à l'état bas

     digitalWrite(clockPin, LOW);

     //on met le bit de donnée courant en place

     digitalWrite(dataPin, le\_bit\_a\_envoyer);

     //enfin on remet l'horloge à l'état haut pour faire prendre en compte ce dernier et finir le cycle

     digitalWrite(clockPin, HIGH);

  }  //et on boucle 8 fois pour faire de même sur chaque bit de l'octet d'ordre

##### Envoyer un char en tant que donnée binaire

Maintenant que l'on a défini une partie de la fonction envoi\_ordre(), il va nous rester un léger problème à régler : envoyer une donnée de type char en tant que suite de bit (ou donnée binaire).

Prenons un exemple : le nombre 231 s'écrit aussi sous la forme 11100111 en base 2 (et oui, c'est le moment de se rappeler [ce que l'on a vu ici](http://sciences.siteduzero.com/tutoriel-3-559277-1-les-bases-du-comptage-2-10-16.html) :-°). Seulement, en voulant envoyer ce nombre sur la broche de donnée pour commander le 74HC595, cela ne marchera pas d'écrire :

digitalWrite(dataPin, 231);

En faisant de cette façon, la carte Arduino va simplement comprendre qu'il faut mettre un état HAUT (car 231 est différent de 0) sur sa broche de sortie que l'on a nommée dataPin. Pour pouvoir donc envoyer ce nombre sous forme binaire, il va falloir ajouter à la fonction que l'on a créé un morceau de code supplémentaire.

Ce que nous allons va faire va être une vraie boucherie : on va découper ce nombre en huit tranches et envoyer chaque morceau un par un sur la sortie dataPin. ^^

Pour découper ce nombre, ça va pas être de la tarte... euh... je m'égare. :-°On va utiliser une technique qui se nomme, tenez-vous bien, le **masquage**. On va en fait utiliser un masque qui va cacher la véritable valeur du nombre 231. Bon bon, je vous explique.

Tout d'abord, on va considérer que le nombre 231 est vu sous sa forme binaire, qui je le rappel est 11100111, par votre carte Arduino. Donc, lorsque l'on va passer en paramètre donnee le nombre 231, le programme verra la suite de 1 et de 0 : 11100111. Jusque-là, rien de bien sorcier.

Voilà donc notre suite de 1 et de 0 que l'on va devoir découper. Alors, il n'existe pas de fonction toute prête spécialement conçue pour découper un nombre binaire. Non, ça va être à nous de faire cela. Et c'est pourquoi je vous parlais du masquage. Cette technique ne porte pas son nom par hasard, en effet, nous allons réellement utiliser un **masque**. Quelques précisions s'imposent, je le sens bien.

Reprenons notre suite binaire :

Image utilisateur

Notre objectif étant d'envoyer chaque bit un par un, on va faire croire à l'Arduino que cette suite n'est composée que d'un seul bit. En clair, on va cacher les 7 autres bits en utilisant un masque :

Image utilisateur

Ce qui, au final, donnera :

Image utilisateur

L'Arduino ne verra donc qu'un seul bit.

Et les autres, il les voit pas, comment on peut envoyer les 8 bits alors ? o_O

Bien sûr, les autres, l'Arduino ne les voit pas. C'est pourquoi l'on va faire évoluer le masque et révéler chaque bit un par un. En faisant cela huit fois, on aura envoyé les 8 bits à la suite :



On peut aussi faire évoluer le masque dans le sens opposé :



L'étape qui suit est donc d'identifier le bit à envoyer en premier. C'est là que rentre en jeu le paramètre sens. On a le choix d'envoyer soit le bit de poids fort (on l'appelle **MSB**, Most Significant Bit) en premier et finir par le bit de poids faible (Less Significant Bit, **LSB**) ; soit dans le sens opposé, du LSB vers le MSB. On parle alors d'envoi MSB First (pour "bit de poids fort en premier") ou LSB First.

À présent, voyons comment appliquer la technique de masquage que je viens de vous présenter

##### Les masques en programmation

Maintenant que vous connaissez cela, nous allons pouvoir voir comment isoler chacun des bits pour les envoyer un par un.

En programmation, il est évident que l'on ne peut pas mettre un masque papier sur les bits pour les cacher. :pIl existe donc un moyen de les cacher. Cela va faire appel à la **logique binaire**. Nous n'entrerons pas dans le détail, mais sachez que nous allons employer des **opérateurs logiques**. Il en existe plusieurs, dont deux très utilisés, même dans la vie courante, l'opérateur **ET** et **OU**.

Commençons par l'opérateur logique ET (je vous laisse regarder le OU tout seul, nous n'en aurons pas besoin ici). Il s'utilise avec le symbole **&** que vous trouverez sous la touche 1 au-dessus de la lettre "a" sur un clavier azerty.

Pour envoyer le premier bit de notre donnée, nous allons effectuer le masquage avec cet opérateur logique dont la table de vérité se trouve être la suivante :

**Table de vérité du ETBit 1 Bit 2 Résultat 0 0 0 0 1 0 1 0 0 1 1 1**

| **Table de vérité du ET** |
| --- |
| **Bit 1** | **Bit 2** | **Résultat** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Je ne comprends pas trop où tu veux en venir ? :euh:

Je vais vous expliquer.

Pour faire le masquage, on va faire une opération avec ce fameux ET logique. Il s'agit de la même chose que si l'on additionnait deux nombres ensemble, ou si on les multipliait. Dans notre cas l'opération est "un peu bizarre". Disons que c'est une opération évoluée.

Cette opération va utiliser deux nombres : le premier on le connaît bien, il s'agit de la suite logique 11100111, quant au second, il s'agira du masque. Pour l'instant, vous ne connaissez pas la valeur du masque, qui sera lui aussi sous forme binaire. Pour définir cette valeur, on va utiliser la table de vérité précédente.

Afin que vous ne vous perdiez pas dans mes explications, on va prendre pour objectif d'envoyer le bit de poids faible de notre nombre 11100111 (celui tout à droite).

Le code qui suit est un pseudo-code, mis sous forme d'une opération mathématique telle que l'on en ferait à l'école :

. 11100111 (donnée à transmettre)

& 00000001 (on veut envoyer uniquement le bit de poids faible)

  --------

  00000001 (donnée à transmettre au final) -> soit 1

Pour comprendre ce qui vient de se passer, il faut se référer à la table de vérité de l'opérateur ET : on sait que lorsque l'on fait 1 et 0 le résultat est 0. Donc, pour cacher tous les bits du nombre à masquer, il n'y a qu'à mettre que des 0 dans le masque. Là, l'Arduino ne verra que le bit 0 puisque le masque aura caché au complet le nombre du départ.  
On sait aussi que 1 ET 1 donne 1. Donc, lorsque l'on voudra montrer un bit à l'Arduino, on va mettre un 1 dans le masque, à l'emplacement du bit qui doit être montré.

Pour monter ensuite le bit supérieur au bit de poids faible, on procède de la même manière :

. 11100111 (donnée à transmettre)

& 00000010 (on veut envoyer uniquement le deuxième bit)

  --------

  00000010 (donnée à transmettre au final) -> soit 1

Pour le quatrième bit en partant de la droite :

. 11100111 (donnée à transmettre)

& 00001000 (on veut envoyer uniquement le quatrième bit)

  --------

  00000000 (donnée à transmettre au final) -> soit 0

Dans le cas où vous voudriez montrer deux bits à l'Arduino (ce qui n'a aucun intérêt dans notre cas, je fais ça juste pour vous montrer) :

. 11100111 (donnée à transmettre)

& 01000100 (on veut envoyer uniquement le quatrième bit)

  --------

  01000100 (donnée à transmettre au final) -> soit 68 en base décimale

##### L'évolution du masque

Ce titre pourrait être apparenté à celui d'un film d'horreur, mais n'indique finalement que nous allons faire évoluer le masque automatiquement à chaque fois que l'on aura envoyé un bit.

Cette fois, cela va être un peu plus simple car nous n'avons qu'à rajouter un opérateur spécialisé dans le décalage. Si l'on veut déplacer le 1 du masque (qui permet de montrer un bit à l'Arduino) de la droite vers la gauche (pour le LSBFirst) ou dans l'autre sens (pour le MSBFirst), nous avons la possibilité d'utiliser l'opérateur **<<** pour décaler vers la gauche ou **>>** pour décaler vers la droite. Par exemple :

.  00000001 (masque initial)

<<        3 (on décale de trois bits)

   --------

   00001000 (masque final, décalé)

Et dans le sens opposé :

.  10000000 (masque initial)

>>        3 (on décale de trois bits)

   --------

   00010000 (masque final, décalé)

Avouez que ce n'est pas très compliqué maintenant que vous maîtrisez un peu les masques. ;)

On va donc pouvoir isoler un par un chacun des bits pour les envoyer au 74HC595. Comme le sens dépend d'un paramètre de la fonction, nous rajoutons un test pour décaler soit vers la droite, soit vers la gauche.  
Voici la fonction que nous obtenons à la fin :

void envoi\_ordre(int dataPin, int clockPin, boolean sens, char donnee)

{

  for(int i=0; i<8; i++) //on va parcourir chaque bit de l'octet

  {

     //on met l'horloge à l'état bas

     digitalWrite(clockPin, LOW);

     //on met le bit de donnée courante en place

     if(sens)

        //envoie la donnée en allant de droite à gauche, en partant d'un masque de type "00000001"

        digitalWrite(dataPin, donnee & 0x01<<i);

     else

        //envoie la donnée en allant de gauche à droite, en partant d'un masque de type "10000000"

        digitalWrite(dataPin, donnee & 0x80>>i);

     //enfin on remet l'horloge à l'état haut pour faire prendre en compte cette dernière

     digitalWrite(clockPin, HIGH);

  }

}

Oula ! Hé ! Stop ! C'est quoi ce 0x01 et ce 0x80 ? Qu'est-ce que ça vient faire là, c'est pas censé être le masque que l'on doit voir ?

Si, c'est bien cela. Il s'agit du masque... écrit sous sa forme hexadécimale. Il aurait été bien entendu possible d'écrire : 0b00000001 à la place de 0x01, ou 0b10000000 à la place de 0x80. On a simplement opté pour la base hexadécimale qui est plus facile à manipuler.

Cette technique de masquage peut sembler difficile au premier abord mais elle ne l'est pas réellement une fois que l'on a compris le principe. Il est essentiel de comprendre comment elle fonctionne pour aller loin dans la programmation de micro-contrôleur (pour paramétrer les registres par exemple), et vous en aurez besoin pour les exercices du chapitre suivant. Pour plus d'informations un bon tuto plus complet mais rapide à lire est [rédigé ici...](http://www.siteduzero.com/tutoriel-3-32351-introduction-aux-operateurs-de-bits.html) en PHP, mais c'est pareil.

##### Un petit programme d'essai

Je vous propose maintenant d'essayer notre belle fonction. Pour cela, quelques détails sont à préciser/rajouter.

Pour commencer, il nous faut déclarer les broches utilisées. Il y en a trois : verrou, horloge et data. Pour ma part elles sont branchées respectivement sur les broches 11, 12 et 10. Il faudra donc aussi les déclarer en sortie dans le setup(). Si vous faites de même vous devriez obtenir le code suivant :

//Broche connectée au ST\_CP du 74HC595

const int verrou = 11;

//Broche connectée au SH\_CP du 74HC595

const int horloge = 12;

//Broche connectée au DS du 74HC595

const int data = 10;

void setup() {

  //On met les broches en sortie

  pinMode(verrou, OUTPUT);

  pinMode(horloge, OUTPUT);

  pinMode(data, OUTPUT);

}

Ensuite, nous allons nous amuser à afficher un nombre allant de 0 à 255 en binaire. Ce nombre peut tenir sur un octet, ça tombe bien car nous allons justement transmettre un octet ! Pour cela, nous allons utiliser une boucle for() allant de 0 à 255 et qui appellera notre fonction.  
Avant cela, je tiens à rappeler qu'il faut aussi mettre en place le verrou en encadrant l'appel de notre fonction. Rappelez-vous, si nous ne le faisons pas, l'affichage risque de scintiller.

//On active le verrou le temps de transférer les données

digitalWrite(verrou, LOW);

//on envoi toutes les données grâce à notre belle fonction (octet inversée avec '~' pour piloter les LED à l'état bas)

envoi\_ordre(data, horloge, 1, ~j);

//et enfin on relâche le verrou

digitalWrite(verrou, HIGH);

Et voici le code complet que vous aurez surement deviné :

//Broche connectée au ST\_CP du 74HC595

const int verrou = 11;

//Broche connectée au SH\_CP du 74HC595

const int horloge = 12;

//Broche connectée au DS du 74HC595

const int data = 10;

void setup() {

  //On met les broches en sortie

  pinMode(verrou, OUTPUT);

  pinMode(horloge, OUTPUT);

  pinMode(data, OUTPUT);

}

void loop() {

  //on affiche les nombres de 0 à 255 en binaire

  for (char i = 0; i < 256; i++) {

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //on envoi toutes les données grâce à notre belle fonction

    envoi\_ordre(data, horloge, 1, ~i);

    //et enfin on relâche le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(1000);

  }

}

void envoi\_ordre(int dataPin, int clockPin, boolean sens, char donnee)

{

  for(int i=0; i<8; i++) //on va parcourir chaque bit de l'octet

  {

     //on met l'horloge à l'état bas

     digitalWrite(clockPin, LOW);

     //on met le bit de donnée courante en place

     if(sens)

        digitalWrite(dataPin, donnee & 0x01<<i);

     else

        digitalWrite(dataPin, donnee & 0x80>>i);

     //enfin on remet l'horloge à l'état haut pour faire prendre en compte cette dernière

     digitalWrite(clockPin, HIGH);

  }

}

Et voila le travail ! :

<http://www.youtube.com/v/if2xNvj7DYo>

#### La fonction magique, ShiftOut

Vous êtes content ? vous avez une belle fonction qui marche bien et fait le boulot proprement ? Alors laissez-moi vous présenter une nouvelle fonction qui s'appelle shiftOut(). Quel est son rôle ? Faire exactement la même chose que la fonction dont l'on vient juste de finir la création.

\*#@"e !! :colere:

Alors oui je sais, c'est pas sympa de ma part de vous avoir fait travailler mais admettez que c'était un très bon exercice de développement non ? À présent vous comprenez comment agit cette fonction et vous serez mieux capable de créer votre propre système que si je vous avais donné la fonction au début en disant : "voilà, c'est celle-là, on l'utilise comme ça, ça marche, c'est beau... mais vous avez rien compris".

Comme je vous le disais précédemment, cette fonction sert à faire ce que l'on vient de créer, mais elle est déjà intégrée à l'environnement Arduino (donc a été testée par de nombreux développeurs, ne laissant pas beaucoup de place pour les bugs !).  
Cette fonction prend quatre paramètres :

* La broche de donnée
* La broche d'horloge
* Le sens d'envoi des données (utiliser avec deux valeurs symboliques, MSBFIRST ou LSBFIRST)
* L'octet à transmettre

Son utilisation doit maintenant vous paraître assez triviale. Comme nous l'avons vu plutôt, il suffit de bloquer le verrou, envoyer la donnée avec la fonction puis relâcher le verrou pour constater la mise à jour des données.  
Voici un exemple de loop avec cette fonction :

void loop()

{

  //on affiche les nombres de 0 à 255 en binaire

  for (int i = 0; i < 256; i++)

  {

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //on envoi toutes les données grâce à shiftOut (octet inversée avec '~' pour piloter les LED à l'état bas)

    shiftOut(data, horloge, LSBFIRST, ~i);

    //et enfin on relache le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(1000);

  }

}

### Exercices : encore des chenillards !

Je vous propose maintenant trois exercices pour jouer un peu avec ce nouveau composant et tester votre habileté au code. Le but du jeu est d'arriver à reproduire l'effet proposé sur chaque vidéo. Le but second est de le faire intelligemment... Autrement dit, tous les petits malins qui se proposeraient de faire un "tableau de motif" contenant les valeurs "affichages binaires" successives devront faire autrement. ;)

Amusez vous bien !

PS : Les corrections seront juste composées du code de la loop avec des commentaires. Le schéma reste le même ainsi que les noms de broches utilisés précédemment.

PPS : La bande son des vidéos est juste là pour cacher le bruit de la télé... je n'y ai pas pensé quand je faisais les vidéos et Youtube ne permet pas de virer la bande audio...

#### "J'avance et repars !"

##### Consigne

Pour ce premier exercice, histoire de se mettre en jambe, nous allons faire une animation simple. Pour cela, il suffit de faire un chenillard très simple, consistant en une LED qui "avance" du début à la fin de la ligne. Arrivée à la fin elle repart au début. Si ce n'est pas clair, regardez la vidéo ci-dessous ! (Éventuellement vous pouvez ajouter un bouton pour inverser le sens de l'animation).

<http://www.youtube.com/v/uMiJnwISEFA>

##### Correction

void loop() {

  for (int i = 0; i < 8; i++) {

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //on envoie la donnée

    //ici, c'est assez simple. On va décaler l'octet 00000001 i fois puis l'envoyer

    shiftOut(data, horloge, LSBFIRST, ~(0x01 << i));

    //et enfin on relache le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(250);

  }

}

#### "J'avance et reviens !"

##### Consigne

Cette seconde animation ne sera pas trop compliquée non plus. La seule différence avec la première est que lorsque la "lumière" atteint la fin de la ligne, elle repart en arrière et ainsi de suite. Là encore si ce n'est pas clair, voici une vidéo :

<http://www.youtube.com/v/kYtlfWnNC34>

##### Correction

Dans cet exercice, le secret est d'utiliser de manière intelligente le paramètre LSBFIRST ou MSBFIRST pour pouvoir facilement inverser le sens de l'animation sans écrire deux fois la boucle for.

char sens = MSBFIRST; //on commence à aller de droite vers gauche

void loop() {

  for (int i = 0; i < 7; i++) { //on ne fait la boucle que 7 fois pour ne pas se répéter au début et à la fin

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //on envoie la donnée

    //On va décaler l'octet 00000001 i fois puis l'envoyer

    shiftOut(data, horloge, sens, ~(0x01 << i));

    //et enfin on relache le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(250);

  }

  sens = !sens; //on inverse le sens d'affichage pour la prochaine fois (MSBFIRST <-> LSBFIRST)

}

#### Un dernier pour la route !

##### Consigne

Pour cette dernière animation, il vous faudra un peu d'imagination. Imaginez le chenillard numéro 1 allant dans les deux sens en même temps... C'est bon ? si non alors voici la vidéo :

<http://www.youtube.com/v/fjev82HNJaQ>

##### Correction

void loop() {

  char donnee = 0;

  for (int i = 0; i < 8; i++) {

    //on saute la boucle si i vaut 4 (pour une histoire de fluidité de l'animation, tester sans et vous verrez)

    if(i == 4)

       continue;

    //calcule la donnée à envoyer

    donnee = 0;

    donnee = donnee | (0x01 << i); // on calcule l'image du balayage dans un sens

    donnee = donnee | (0x80 >> i); // et on ajoute aussi l'image du balayage dans l'autre sens

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //on envoie la donnée

    shiftOut(data, horloge, LSBFIRST, ~donnee);

    //et enfin on relache le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(250);

  }

}

#### Exo bonus

##### Consigne

Ici le but du jeu sera de donner un effet de "chargement / déchargement" en alternance...  
Comme d'habitude, voici la vidéo pour mieux comprendre...

<http://www.youtube.com/v/1h0kh57GVps>

##### Correction

Dans cet exercice, tout repose sur l'utilisation du MSBFIRST ou LSBFIRST ainsi que du complément appliqué sur la donnée. Ce dernier permet d'activer ou non les LEDs et le premier atout permet d'inverser l'effet.

char extinction = 0; //on commence à aller de droite vers gauche

void loop() {

  char donnee = extinction; //on démarre à 0 ou 1 selon...

  for (int i = 0; i < 8; i++) {

    //On active le verrou le temps de transférer les données

    digitalWrite(verrou, LOW);

    //si on est en train d'éteindre

    if(extinction)

      shiftOut(data, horloge, MSBFIRST, ~donnee);//on envoie la donnée inversé

    //sinon

    else

      shiftOut(data, horloge, LSBFIRST,  donnee);//on envoie la donnée normale

    //et enfin on relache le verrou

    digitalWrite(verrou, HIGH);

    //une petite pause pour constater l'affichage

    delay(250);

    donnee = donnee | (0x01 << i); //et on met à jour la donnée en cumulant les décalages

  }

  extinction = !extinction; //permet d'inverser "MSBFIRST <-> LSBFIRST" comme dans l'exercice 2

}

### Pas assez ? Augmenter encore !

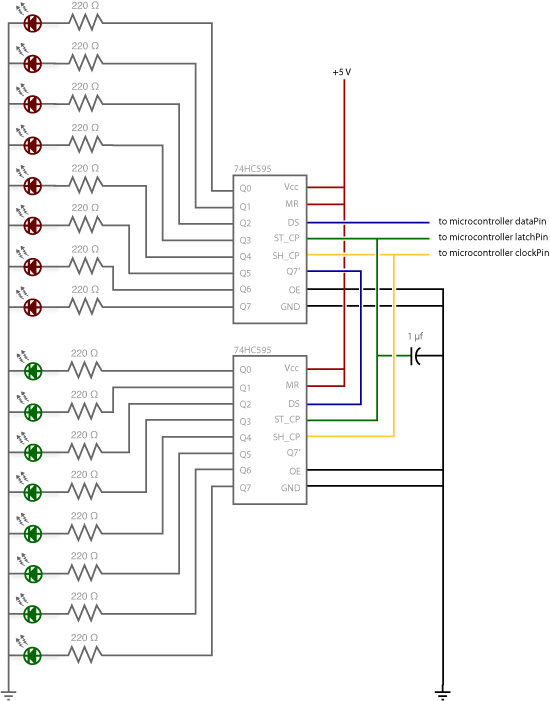
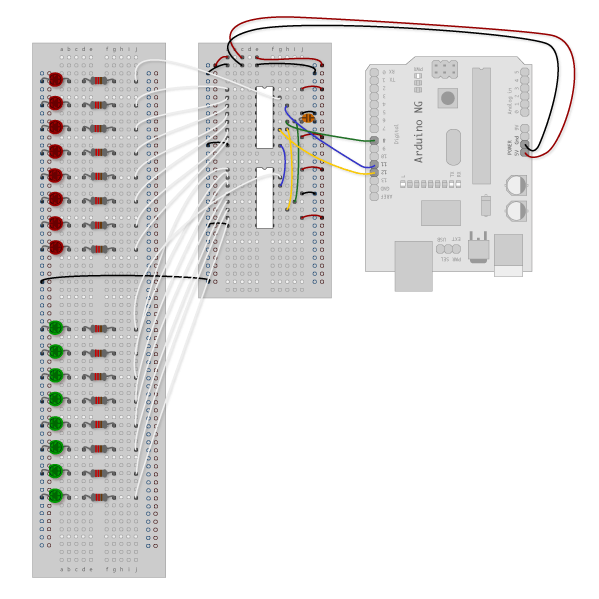
Si jamais 8 nouvelles sorties ne vous suffisent pas (bien que cela n'en face que 5 au total puisque trois sont prises pour communiquer avec le composant), les ingénieurs ont déjà tout prévu. Ainsi il est possible de mettre en cascade plusieurs 74HC595 !

Pour cela, le 595 dispose d'une broche appelée "débordement". Lorsque vous envoyez un seul octet au 74HC595, rien ne se passe sur cette broche. Cependant, si vous envoyez plus d'un octet, les huit derniers bits seront conservés par le composant, tandis que les autres vont être "éjectés" vers cette fameuse sortie de débordement (numéro 9). Le premier bit envoyé ira alors vers le 74HC595 le plus loin dans la chaine. Souvenez-vous, elle s'appelle "serial data output" et j'avais dit qu'on reviendrait dessus. D'une manière très simple, les bits éjectés vont servir aux éventuels 74HC595 qui seront mis en aval de celui-ci.

##### Branchement

Il suffit dons de mettre deux 595 bout-à-bout en reliant la broche de débordement du premier sur la broche de donnée du second. Ainsi, les bits "en trop" du premier arriveront sur le second. Afin que le second fonctionne, il faut aussi également relier les mêmes broches pour l'horloge et le verrou (reliées en parallèle entre les deux).

Les images proviennent d'une [explication du site Arduino](http://arduino.cc/en/Tutorial/ShiftOut). Attention, dans ce schéma les LEDs sont branchées "à l'envers" de ce que nous avons l'habitude de faire.

Deux 595 en cascade, schémaDeux 595 en cascade, breadboard

##### Exemple d'un affichage simple

Au niveau du programme, il suffira de faire appel deux fois de suite à la fonction shiftOut pour tout envoyer (2 fois 8 bits). Ces deux appels seront encadrés par le verrou pour actualiser l'affichage des données. On commence par envoyer la donnée qui doit avancer le plus pour atteindre le second 595, puis ensuite on fait celle qui concerne le premier 595.  
Voici un exemple :

const int verrou = 11;

const int donnee = 10;

const int horloge = 12;

char premier = 8; //en binaire : 00001000

char second = 35; //en binaire : 00100011

void setup()

{

   //on déclare les broches en sortie

   pinMode(verrou, OUTPUT);

   pinMode(donnee, OUTPUT);

   pinMode(horloge, OUTPUT);

   //puis on envoie les données juste une fois

   //on commence par mettre le verrou

   digitalWrite(verrou, LOW);

   //on envoie la seconde donnée d'abord

   shiftOut(donnee, horloge, LSBFIRST, ~second); //les LEDs vertes du montage

   //on envoie la première donnée

   shiftOut(donnee, horloge, LSBFIRST, ~premier); //Les LEDs rouges du montage

   //et on relache le verrou pour mettre à jour les données

   digitalWrite(verrou, HIGH);

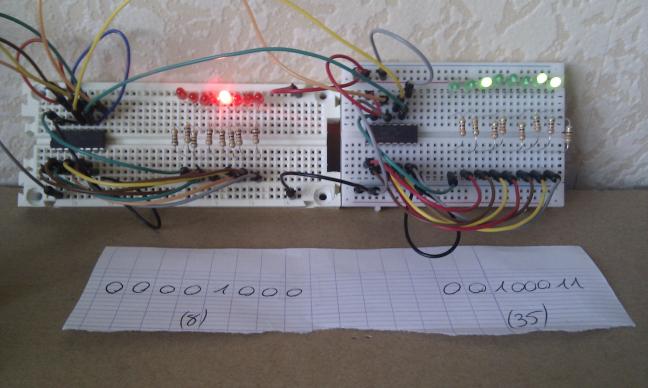
}

void loop()

{

   //rien à faire

}



##### Exemple d'un chenillard

Voici maintenant un petit exemple pour faire un chenillard sur 16 LEDs. Pour cela, j'utiliserai un int qui sera transformé en char au moment de l'envoi. Il faudra donc le décaler vers la droite de 8 bits pour pouvoir afficher ses 8 bits de poids fort. Voici une loop pour illustrer mes propos (le setup étant toujours le même).

void loop()

{

   int masque = 0;

   for(int i=0; i<16; i++)

   {

     masque = 0x01 << i; //on décale d'un cran le masque

     //on commence par mettre le verrou

     digitalWrite(verrou, LOW);

     //on envoie la seconde donnée d'abord

     shiftOut(donnee, horloge, LSBFIRST, ~(masque & 0x00FF)); //On envoie les 8 premiers bits

     //on envoie la première donnée

     shiftOut(donnee, horloge, LSBFIRST, ~((masque & 0xFF00) >> 8)); //On envoie les 8 derniers bits

     //et on relache le verrou pour mettre à jour les données

     digitalWrite(verrou, HIGH);

     delay(500);

   }

}

<http://www.youtube.com/v/in_GIqS5CU4>

Ce composant peut vous paraître un peu superflu mais il existe en fait de très nombreuses applications avec. Par exemple, si vous voulez réaliser un cube de LED (disons 4x4x4 pour commencer gentiment). Si vous vouliez donner une broche par LED vous seriez bloquer puisque Arduino n'en possède pas autant (il vous en faudrait 32). Ici le composant vous permet donc de gérer plus de sorties que vous ne le pourriez initialement.

On achève enfin cette deuxième partie où vous avez pu acquérir un ensemble de connaissances nécessaires pour poursuivre la lecture de ce tutoriel. La prochaine partie va traiter sur la communication entre une Arduino et un ordinateur ou même entre deux Arduino. Cela risque d'être prometteur !

# <http://www.pobot.org/Piloter-des-afficheurs-7-segments.html>

## Piloter des afficheurs 7 segments

samedi 18 octobre 2008, par [Julien Holtzer](http://www.pobot.org/_Julien-Holtzer_.html)

* [Exemples et cahier des charges](http://www.pobot.org/Piloter-des-afficheurs-7-segments.html#exemples_et_cahier_des_charges)
* [Quelques solutions possibles](http://www.pobot.org/Piloter-des-afficheurs-7-segments.html#quelques_solutions_possibles)
* [Réalisation du chronomètre](http://www.pobot.org/Piloter-des-afficheurs-7-segments.html#realisation_du_chronometre)
* [Code source](http://www.pobot.org/Piloter-des-afficheurs-7-segments.html#code_source)

Les afficheurs 7-segments sont des chiffres lumineux que l’on peut piloter pour choisir les valeurs successives qui vont s’allumer. Les 10 chiffres (de 0 à 9) sont décomposés en segments : chacun de ces segments peut s’allumer indépendamment des autres.

[](http://www.pobot.org/IMG/png/afficheurs_7seg.png)

**Afficheurs 7-segment**

On peut représenter l’afficheur comme autant de leds que de segments, mais pour limiter le nombre de pattes, ces leds sont reliées par une même patte. Une diode, même électro-luminescente, étant un bipole polarisé, on distingue deux types d’afficheur : ceux dont la cathode est commune (abbréviation CC) et ceux à anode commune (AC). D’autres informations [ici](http://www.discip.ac-caen.fr/sti/stibacs/modules.php?name=News&file=article&sid=92)

Anecdote rigolote, il n’y a pas toujours 7 segments mais plutôt 8 car un . est présent sur la plupart d’entre eux, pour servir de virgule par exemple. Bien sûr il n’est pas nécessaire de le câbler, comme il n’est pas nécessaire de câbler tous les segments si vous ne comptez utiliser qu’une partie d’entre eux (pour un 1 par exemple) puisque chaque led est indépendante.

### Exemples et cahier des charges

On trouve des afficheurs 7-segments sur un radio-réveil, un chronomètre géant, etc. Une utilisation courante en robotique est d’afficher en permanence la tension disponible dans le robot grâce à un voltmètre simplifié.

Nous présentons dans cet article la réalisation d’un chronomètre pour nos courses de robots Ma-vin. [A CONTINUER]

### Quelques solutions possibles

**La problématique**

Deux aspects :   
- le nombre de pattes à contrôler : jusqu’à 9 par afficheur, sachant qu’on peut multiplexer les afficheurs pour avoir 8 pattes communes pour chacun des segments et 1 patte par afficheur. Cela peut s’avérer trop important si on a un microcontrôleur limité ou qui doit gérer d’autres fonctions que l’affichage  
- la puissance disponible : le microcontrôleur délivre un courant limité sur chacune des pattes, alors que l’afficheur va demander plus que disponible, surtout s’il est de grande taille

**Une solution simple**

Utiliser des transistors [A CONTINUER]

**Une solution à circuits intégrés**

Utiliser un transcodeur dédié aux afficheurs 7 segments. Il s’agit d’un circuit intégré qui va réduire le nombre de sorties nécessaires sur le microcontrôleur et/ou qui va intégrer les transistors pour limiter le nombre de composants.

**Une nouvelle solution**

Pour notre problème de display 7 segments, et histoire de nous simplifier la vie, Eric nous suggère de jeter un oeil sur ceci : [http://www.maxim-ic.com/quick\_view2...](http://www.maxim-ic.com/quick_view2.cfm/qv_pk/1339).

[PDF - 734.5 ko](http://www.pobot.org/IMG/pdf/MAX7219-MAX7221.pdf)

**Datasheet MAX7219**

**Une autre approche**

Il existe bien sûr une solution "tout-en-un" où même les afficheurs sont intégrés. C’est le plus simple question mise en oeuvre :  
[board SDP8](http://www.futurlec.com/DevBoardAccessories.shtml).

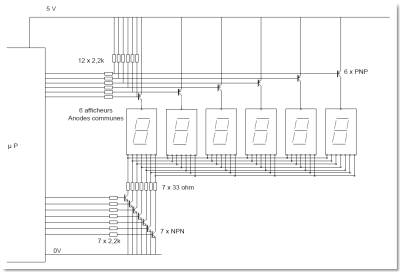


Il y a 8 afficheurs 7 segments, ça se commande avec 4 sorties du microcontrôleur, et la bufferisation est assurée par le chip Maxim présenté ci-dessus.

Cette solution est très intéressante et économique (en argent et en temps) mais elle ne nous convient pas car nous voulons utiliser de grands afficheurs.

### Réalisation du chronomètre

Nous utiliserons des transistors, à la fois pour sélectionner successivement chaque afficheur, et pour positionner les segments à éclairer pour afficher la valeur souhaitée.

[](http://www.pobot.org/IMG/pdf/afficheurs.pdf)

**Schéma de cablage des afficheurs**

Il y a donc besoin de :   
- 6 afficheurs 7 segments à anode commune  
- 6 transistors PNP pour la sélection successive des afficheurs  
- 7 transistors NPN pour la sélection du chiffre à afficher  
- 7 résistances 33 Ohms (voir calculs pour la luminosité)  
- 19 résistances 2,2 kOhms

### Code source

/\*  
Cible : ATMega8  
Quartz : 8MHz  
Compilateur : avr-gcc (WInAVR)  
\*/

#include   
#include

// compteur du nombre d’interruptions timer  
volatile int tmr\_cnt = 0 ;

// vu le prescaling et l’étendue du compteur, un intervalle de 100ms correspond à 3 interruptions  
#define \_100\_MS (3 \* 64)  
#define \_1\_S (30 \* 4)

#define DIGIT\_0 0b00111111  
#define DIGIT\_1 0b00000110  
#define DIGIT\_2 0b01011011  
#define DIGIT\_3 0b01001111  
#define DIGIT\_4 0b01100110  
#define DIGIT\_5 0b01101101  
#define DIGIT\_6 0b01111101  
#define DIGIT\_7 0b00000111  
#define DIGIT\_8 0b01111111  
#define DIGIT\_9 0b01101111  
#define DIGIT\_A 0b01110111  
#define DIGIT\_B 0b01111100  
#define DIGIT\_b 0b01111100  
#define DIGIT\_C 0b00111001  
#define DIGIT\_d 0b01011110  
#define DIGIT\_D 0b01011110  
#define DIGIT\_E 0b01111001  
#define DIGIT\_F 0b01110001  
#define DIGIT\_g 0b01111101  
#define DIGIT\_H 0b01110110  
#define DIGIT\_h 0b01110100  
#define DIGIT\_L 0b00111000  
#define DIGIT\_l DIGIT\_1  
#define DIGIT\_O DIGIT\_0  
#define DIGIT\_o 0b01011100  
#define DIGIT\_r 0b01010000  
#define DIGIT\_W 0b00111110  
#define DIGIT\_w 0b00011100  
#define DIGIT\_sp 0b00000000  
#define DIGIT\_dp 0b10000000

unsigned char digits[] =  *DIGIT\_0,  
DIGIT\_1,  
DIGIT\_2,  
DIGIT\_3,  
DIGIT\_4,  
DIGIT\_5,  
DIGIT\_6,  
DIGIT\_7,  
DIGIT\_8,  
DIGIT\_9,  
DIGIT\_A,  
DIGIT\_B,  
DIGIT\_C,  
DIGIT\_D,  
DIGIT\_E,  
DIGIT\_F* ;

unsigned char msg[] =  *DIGIT\_H,  
DIGIT\_E,  
DIGIT\_L,  
DIGIT\_L,  
DIGIT\_O,  
DIGIT\_sp,  
DIGIT\_W,  
DIGIT\_o,  
DIGIT\_r,  
DIGIT\_l,  
DIGIT\_d,  
DIGIT\_dp,  
DIGIT\_dp,  
DIGIT\_dp,  
DIGIT\_sp,  
DIGIT\_sp,  
DIGIT\_sp,  
DIGIT\_sp* ;

unsigned char \*p = msg ;

unsigned char digit\_num = 0 ;  
unsigned char digit\_sel ;  
unsigned char digit\_value[4] = *DIGIT\_sp, DIGIT\_sp, DIGIT\_sp, DIGIT\_sp* ;

/\*  
Interrupt handler pour l’overflow du timer 0 (T=33ms)  
\*/  
SIGNAL (SIG\_OVERFLOW0)  *// A chaque interruption, on allume un des digits en activant les segments correspondants  
// au motif qu’il doit reproduire via le port A et en activant sa cathode via le port B.  
// Vue la fréquence de balayage, l’oeil a l’illusion que les 4 digits sont allumés en permanence  
PORTA = digit\_value[digit\_num] ;  
PORTB = digit\_sel ;*

*digit\_num = (digit\_num + 1) & 0x03 ;  
digit\_sel = 1 << digit\_num ;*

*// si on est arrivé au nombre nécessaire à l’obtention du délai de défilement souhaité,  
// on modifie le contenu du buffer représentant le message à afficher  
if (tmr\_cnt == 0)   
digit\_value[3] = digit\_value[2] ;  
digit\_value[2] = digit\_value[1] ;  
digit\_value[1] = digit\_value[0] ;  
digit\_value[0] = \*p++ ;*

*if (p - msg == sizeof(msg)) p = msg ;*

*// mise à jour du compteur d’interruptions pour la gestion du défilement  
tmr\_cnt = (tmr\_cnt + 1) % (2 \* \_1\_S) ;*

/\*  
Initialisation des ports  
\*/  
void port\_init(void)  *DDRA = 0xFF ;  
DDRB = 0xFF ;*

/\*  
Initialisation des périphériques  
\*/  
void init\_devices(void)  *cli() ; // inhibition des interruptions (pour avoir la paix)  
port\_init() ; // initialisation de prts*

*// MCU Control Register  
// - interruptions externes sur niveau bas de la ligne  
// - sleep mode désactivé  
MCUCR = 0x00 ;*

*// INT0 et INT1 inactives  
GICR = 0x00 ;*

*// activation de l’interrupt d’overflow du timer 0  
TIMSK = 0x01 ;*

*// réglage du pre-scaler du timer 0  
TCCR0 = 0x03 ;*

*sei() ; // autorisation des interruptions*

//- --------------------------------------------------------------------------------  
int main(void)  *init\_devices() ;*

*// rien à ajouter : tout est fait dans l’interrupt handler  
return 0 ;*

# http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html

## Ecran CLCDxxx sur I2C avec Arduino

samedi 22 octobre 2011, par [Julien Holtzer](http://www.pobot.org/_Julien-Holtzer_.html)

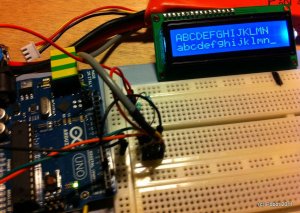
[Lextronic](http://www.pobot.org/+-Lextronic-+.html) propose des [écrans à cristaux liquides LCD économiques](http://www.lextronic.fr/R1576-lcd-i2c.html) déclinés en plusieurs tailles (CLCD162 ou CLCD204) et plusieurs éclairages (vert, bleu et même RGB). Contrairement à d’autres écrans compatibles avec la bibliothèque native *Liquid Crystal* d’[Arduino](http://www.pobot.org/+-Arduino-+.html), nous n’avons pas trouvé de code source compatible avec les commandes I2C de cette gamme. C’est désormais chose faite !

* [L’écran](http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html#l_ecran)
* [Le code](http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html#le_code)
* [Le résultat](http://www.pobot.org/Ecran-CLCDxxx-sur-I2C-avec-Arduino.html#le_resultat)

**Mise à jour** : la bibliothèque est désormais disponible pour le logiciel Arduino 1.0, car les fonctions I2C ont changé ainsi que la classe mère Print (voir notre article complet sur le sujet).

### L’écran

Comme d’autres écrans LCD / I2C, un écran classique à connexion parallèle est soudé sur une petite carte comportant une puce (ici, un AVR Atmel reprogrammable) et un ou plusieurs connecteurs.

[](http://www.pobot.org/IMG/jpg/LCD-I2C_3.jpg)

Le rétro-éclairage de cette version est réglable par un potentiomètre miniature.

[](http://www.pobot.org/IMG/jpg/LCD-I2C_2.jpg)

### Le code

Chaque commande du protocole LCD I2C est composée d’octets à envoyer avec la bibliothèque Wire. Pour simplifier ces commandes, et revenir à des fonctions standards (on parle d’API), une bibliothèque pour Arduino a été écrite selon le format LCDXXX décrit sur le [Playground Arduino](http://www.arduino.cc/playground/Code/LCDAPI).

[Zip - 3.3 ko](http://www.pobot.org/IMG/zip/CLCD.zip)

**Library Arduino "CLCD"**

version pour Arduino 0022 et précédentes

[Zip - 3.4 ko](http://www.pobot.org/IMG/zip/CLCD_pour_Arduino_1.0.zip)

**Library Arduino "CLCD" 1.0**

Mise à jour pour le logiciel Arduino 1.0

### Le résultat

Voici le test disponible dans l’exemple :

[](http://www.pobot.org/IMG/jpg/LCD-I2C_4.jpg)

Et voici la réponse d’Yves, notre lecteur attentif qui a réussit à prendre en main la bibliothèque.

[](http://www.pobot.org/IMG/jpg/LCD-I2C_5.jpg)

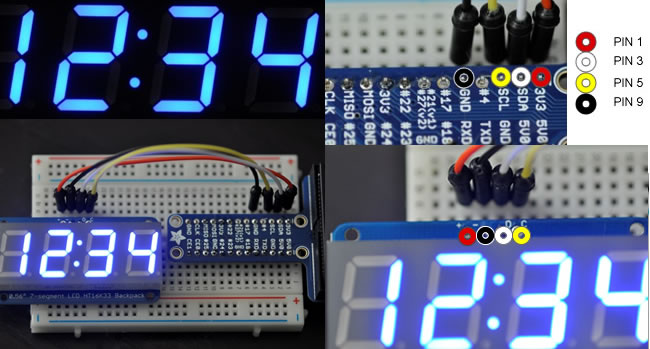
# <http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/>

## RaspberryPI: Afficheur 7 segments avec I2C

* [1 Activation de I2C](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Activation_de_I2C)
  + [1.1 Installation des paquets](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Installation_des_paquets)
  + [1.2 Activation des modules linux](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Activation_des_modules_linux)
  + [1.3 Vérification](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Vrification)
  + [1.4 Animation de test](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Animation_de_test)
    - [1.4.1 Récupération des sources](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Rcupration_des_sources)
    - [1.4.2 Test d’affichage sur la matrice](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Test_drsquoaffichage_sur_la_matrice)
  + [1.5 Pour aller plus loin](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Pour_aller_plus_loin)
    - [1.5.1 Afficher 12:34](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Afficher_1234)
    - [1.5.2 Afficher °C](http://diy.enna.fr/2013/07/16/raspberrypi-afficheur-7-segments-et-i2c/#Afficher_C)

Pour la suite du projet [Pirror](http://diy.enna.fr/pirror-le-miroir-connecte/), nous allons voir comment utiliser I2C pour utiliser un afficheur 7 segments 4 digits d’Adafruit.

Voici le ce qu’on va faire ainsi que le câblage:

[](http://diy.enna.fr/wp-content/uploads/2013/07/7segmentdigit.jpg)

Si vous avez déjà activé I2C, vous pouvez passer ce chapitre.

Tout d’abord, I2C signifie Inter Integrated Circuit. C’est un bus de données normalisé. Il permet à plusieurs chips et modules de communiquer entre eux, très simplement.

En gros, chaque module possède son adresse. Le controleur (RPI ici) peut ainsi s’adresser à un ou plusieurs modules. Les modules peuvent également en mode « clone » en utilisant le même identifiant.Le plus simple, c’est d’aller voir plus en détails…

Je ne connaissais pas du tout la manière dont cela pouvait fonctionner. Vous serez étonné de la facilité d’utilisation de ces modules de LED (ou autres).

Cette page est très clairement inspirée du site suivant:

http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

# Activation de I2C

On active le driver de la puce bcm2708 en ajoutant deux lignes (8 et 9) au fichier « modules »

|  |  |
| --- | --- |
| **sudo** **vi** **/**etc**/**modules | |
| 1  2  3  4  5  6  7  8  9 | *# /etc/modules: kernel modules to load at boot time.*  *#*  *# This file contains the names of kernel modules that should be loaded*  *# at boot time, one per line. Lines beginning with "#" are ignored.*  *# Parameters can be specified after the module name.*    snd-bcm2835  i2c-bcm2708i2c-dev | |

Pour que la modification soit prise en charge, il faut redémarrer le Rpi:

|  |
| --- |
| **sudo** reboot |

## Installation des paquets

On installe ensuite les paquets suivants pour utiliser des scripts python:

|  |
| --- |
| **sudo** **apt-get install** python-smbus i2c-tools |

## Activation des modules linux

On active les modules linux suivants, en retirant les « # » :

|  |  |
| --- | --- |
| **sudo** **vi** **/**etc**/**modprobe.d**/**raspi-blacklist.conf | |
| 1  2  3  4 | *# blacklist spi and i2c by default (many users don't need them)*    blacklist spi-bcm2708blacklist i2c-bcm2708 | |

## Vérification

Ca y est, la matrice de LED peut être branchée comme indiqué sur l’image au dessus.

Par défaut son adresse est en 0x70.

Pour tester si les modules électroniques sont bien reconnus, exécutons:

|  |
| --- |
| **sudo** i2cdetect -y 1 |

(-y 1 correspond au numéro de bus, les Rpi plus anciens sont en 0, les plus récents en 1.)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 0 1 2 3 4 5 6 7 8 9 a b c d e f  00: -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: 70 -- -- -- -- -- -- -- |

On voit ici la matrice de LED en ligne 70: numéro 70.

## Animation de test

### Récupération des sources

|  |  |
| --- | --- |
| 1  2  3 | **sudo** **apt-get install** **git**  **git clone** https:**//**github.com**/**adafruit**/**Adafruit-Raspberry-Pi-Python-Code.git  **cd** Adafruit-Raspberry-Pi-Python-Code**/**Adafruit\_LEDBackpack |

### Test d’affichage sur la matrice

|  |
| --- |
| **sudo** python ex\_7segment\_clock.py  Press CTRL+Z to **exit** |

A ce stade, le script ne vous rend pas la main, et vous devriez voir l’heure de votre RPI ainsi que le séparateur Heures:Minutes, clignoter toutes les secondes.

## Pour aller plus loin

 Nous allons voir comment faire activer n’importe quel segment de l’afficheur.  
Je n’ai pas réussi à trouver des informations pertinentes sur le sujet, ou alors elles n’étaient pas accessible à mon pauvre niveau d’électronicien ;-)… c’est pourquoi je vous partage mes recherches.

 Voici comment est codé l’afficheur, les explications sont en dessous.

[](http://diy.enna.fr/wp-content/uploads/2013/07/7_segments.jpg)

 Chaque chiffre (ou digit – ici au nombre de 4) est décomposé en segments, dans notre cas, 7.  
 On observe également le point (code décimal 128) qui sépare chaque chiffre. Les « : » de l’horloge (séparateur heures-minutes), ne peuvent pas être dissociés, ou les deux sont allumés, ou les deux sont éteints (digit #2).

Trêve de bavardage, voici des cas pratiques:

### Afficher 12:34

Je souhaite afficher l’heure 12:34, il n’y a rien de plus simple.  
En effet, la bibliothèque que l’on utilise, implémente l’affichage d’un caractère [0-9] ou [A-Z].  
Alors pour faire au plus simple:

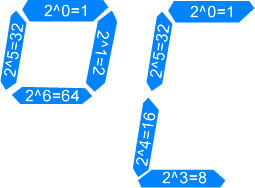
|  |  |
| --- | --- |
| **vi** 4\_7segments.py | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | *#!/usr/bin/python*  *# ===========================================================================*  *# diy.enna.fr - 4\_7segments.py - 2013/07/16*  *# Affichage de 12:34 sur l'afficheur 7 segments*  *# ===========================================================================*    import **time**  import datetime  from Adafruit\_7Segment import SevenSegment    *#Adresse par defaut 0x70.*  segment = SevenSegment**(**address=0x70**)**    *#temps de pause entre l'allumage de chaque segment:*  t=2    segment.writeDigit**(**0,1 **)**  time.sleep**(**t**)**  segment.writeDigit**(**1,2 **)**  time.sleep**(**t**)**    segment.writeDigit**(**2,1 **)**  time.sleep**(**t**)**    segment.writeDigit**(**3,3 **)**  time.sleep**(**t**)**    segment.writeDigit**(**4,4 **)** | |

Exécutez le code avec:

|  |
| --- |
| **sudo** python 4\_7segments.py |

Vous verrez chaque digit s’allumer.

### Afficher °C

OK, mais c’est trop simple alors, je veux afficher ca : (pratique pour suffixer une température non? ;-))  
[](http://diy.enna.fr/wp-content/uploads/2013/07/degesC.jpg)

Une méthode différente permet d’utiliser les codes décimaux représentant les segments. Vous allez voir, ce n’est pas super compliqué:

Pour afficher « degrés Celcius », il nous faut la valeur decimale de « ° » et celle de « C »:

°=1+2+64+32 => 99  
C=1+32+16+8 => 57

Cette fois, on fait appel à « writeDigitRaw » et non pas « writeDigit »

On affiche le tout dans les deux digit de droite (#3 et #4)

|  |  |
| --- | --- |
| **vi** 5\_7segments\_degres.py | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | *#!/usr/bin/python*  *# ===========================================================================*  *# diy.enna.fr - 5\_7segments\_degres.py - 2013/07/16*  *# Affichage degré Celcius sur l'afficheur 7 segments*  *# ===========================================================================*    import **time**  import datetime  from Adafruit\_7Segment import SevenSegment    *#Adresse par defaut 0x70.*  segment = SevenSegment**(**address=0x70**)**    segment.writeDigitRaw**(**3, 99 **)**  segment.writeDigitRaw**(**4, 57**)** | |

Exécutez le code avec:

|  |
| --- |
| **sudo** python 5\_7segments\_degres.py |

Vous devriez voir le signe « °C » s’afficher sur les deux caractères à droite de l’afficheur.

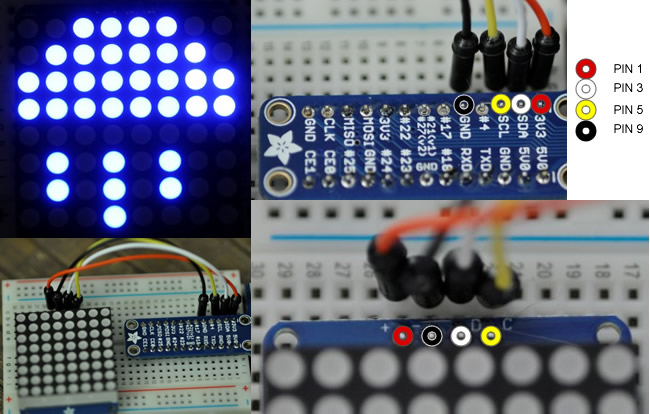
# http://diy.enna.fr/2013/07/15/raspberrypi-i2c/

## RaspberryPI: Matrice de LED avec I2C

* [1 Activation de I2C](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Activation_de_I2C)
  + [1.1 Installation des paquets](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Installation_des_paquets)
  + [1.2 Activation des modules linux](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Activation_des_modules_linux)
  + [1.3 Vérification](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Vrification)
  + [1.4 Animation de test](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Animation_de_test)
    - [1.4.1 Récupération des sources](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Rcupration_des_sources)
    - [1.4.2 Test d’affichage sur la matrice](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Test_drsquoaffichage_sur_la_matrice)
    - [1.4.3 Afficher un nuage](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Afficher_un_nuage)
  + [1.5 Pour aller plus loin](http://diy.enna.fr/2013/07/15/raspberrypi-i2c/#Pour_aller_plus_loin)

Pour la suite du projet [Pirror](http://diy.enna.fr/pirror-le-miroir-connecte/), nous allons voir comment utiliser I2C pour utiliser une matrice de LED 8×8.

Voici le ce qu’on va faire ainsi que le câblage:

[](http://diy.enna.fr/wp-content/uploads/2013/06/nuage_matrice8x8.jpg)

Tout d’abord, I2C signifie Inter Integrated Circuit. C’est un bus de données normalisé. Il permet à plusieurs chips et modules de communiquer entre eux, très simplement.

En gros, chaque module possède son adresse. Le controleur (RPI ici) peut ainsi s’adresser à un ou plusieurs modules. Les modules peuvent également en mode « clone » en utilisant le même identifiant.Le plus simple, c’est d’aller voir plus en détails…

Je ne connaissais pas du tout la manière dont cela pouvait fonctionner. Vous serez étonné de la facilité d’utilisation de ces modules de LED (ou autres).

Cette page est très clairement inspirée du site suivant:

http://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

# Activation de I2C

On active le driver de la puce bcm2708 en ajoutant deux lignes (8 et 9) au fichier « modules »

|  |  |
| --- | --- |
| **sudo** **vi** **/**etc**/**modules | |
| 1  2  3  4  5  6  7  8  9 | *# /etc/modules: kernel modules to load at boot time.*  *#*  *# This file contains the names of kernel modules that should be loaded*  *# at boot time, one per line. Lines beginning with "#" are ignored.*  *# Parameters can be specified after the module name.*    snd-bcm2835  i2c-bcm2708i2c-dev | |

Pour que la modification soit prise en charge, il faut redémarrer le Rpi:

|  |
| --- |
| **sudo** reboot |

## Installation des paquets

On installe ensuite les paquets suivants pour utiliser des scripts python:

|  |
| --- |
| **sudo** **apt-get install** python-smbus i2c-tools |

## Activation des modules linux

On active les modules linux suivants, en retirant les « # » :

|  |  |
| --- | --- |
| **sudo** **vi** **/**etc**/**modprobe.d**/**raspi-blacklist.conf | |
| 1  2  3  4 | *# blacklist spi and i2c by default (many users don't need them)*    blacklist spi-bcm2708blacklist i2c-bcm2708 | |

## Vérification

Ca y est, la matrice de LED peut être branchée comme indiqué sur l’image au dessus.

Par défaut son adresse est en 0x70.

Pour tester si les modules électroniques sont bien reconnus, exécutons:

|  |
| --- |
| **sudo** i2cdetect -y 1 |

(-y 1 correspond au numéro de bus, les Rpi plus anciens sont en 0, les plus récents en 1.)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | 0 1 2 3 4 5 6 7 8 9 a b c d e f  00: -- -- -- -- -- -- -- -- -- -- -- -- --  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --  70: 70 -- -- -- -- -- -- -- |

On voit ici la matrice de LED en ligne 70: numéro 70.

## Animation de test

### Récupération des sources

|  |  |
| --- | --- |
| 1  2  3 | **sudo** **apt-get install** **git**  **git clone** https:**//**github.com**/**adafruit**/**Adafruit-Raspberry-Pi-Python-Code.git  **cd** Adafruit-Raspberry-Pi-Python-Code**/**Adafruit\_LEDBackpack |

### Test d’affichage sur la matrice

|  |
| --- |
| **sudo** python ex\_8x8\_pixels.py  Press CTRL+Z to **exit** |

A ce stade, le script ne vous rend pas la main, et vous devriez voir toutes les LED de la matrice, s’allumer successivement.

La prochaine étape consiste à créer un script python qui va afficher un nuage.

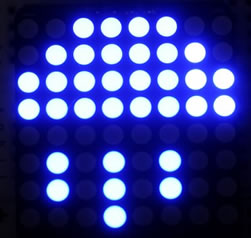
### Afficher un nuage

Copiez-collez le code suivant (qui d’ailleurs et hyper simpliste et pas du tout optimisé :-)dans un fichier 3\_pluie.py).

|  |  |
| --- | --- |
| **vi** 3\_pluie.py | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | *#!/usr/bin/python*  import **time**  import datetime  from Adafruit\_8x8 import EightByEight  *# ===========================================================================*  *# diy.enna.fr - 3\_pluie.py - 2013/07/15*  *# Affichage d'un nuage sur une matrice de LED 8x8 d'Adafruit*  *# ===========================================================================*  *#Adresse par defaut 0x70.*  grid = EightByEight**(**address=0x70**)**    *#On efface la grille*  grid.clear**()**    *#On allume chaque diode une par une avec ses coordonnées.*  grid.setPixel**(**0, 2**)**  grid.setPixel**(**0, 3**)**  grid.setPixel**(**0, 4**)**  grid.setPixel**(**0, 5**)**  grid.setPixel**(**1, 1**)**  grid.setPixel**(**1, 2**)**  grid.setPixel**(**1, 3**)**  grid.setPixel**(**1, 4**)**  grid.setPixel**(**1, 5**)**  grid.setPixel**(**1, 6**)**  grid.setPixel**(**2, 0**)**  grid.setPixel**(**2, 1**)**  grid.setPixel**(**2, 2**)**  grid.setPixel**(**2, 3**)**  grid.setPixel**(**2, 4**)**  grid.setPixel**(**2, 5**)**  grid.setPixel**(**2, 6**)**  grid.setPixel**(**2, 7**)**  grid.setPixel**(**3, 0**)**  grid.setPixel**(**3, 1**)**  grid.setPixel**(**3, 2**)**  grid.setPixel**(**3, 3**)**  grid.setPixel**(**3, 4**)**  grid.setPixel**(**3, 5**)**  grid.setPixel**(**3, 6**)**  grid.setPixel**(**3, 7**)**  grid.setPixel**(**5, 2**)**  grid.setPixel**(**5, 4**)**  grid.setPixel**(**5, 6**)**  grid.setPixel**(**6, 2**)**  grid.setPixel**(**6, 4**)**  grid.setPixel**(**6, 6**)**  grid.setPixel**(**7, 4**)** | |

L’exécution du script

|  |
| --- |
| **sudo** python 3\_pluie.py |

Affichera la matrice suivante:  
[](http://diy.enna.fr/wp-content/uploads/2013/07/nuage.jpg)

## Pour aller plus loin

Pour faire défiler un texte, utilisez cette bibliothèque modifiée:

https://github.com/cameraready/Adafruit-Raspberry-Pi-Python-Code

# http://arduino103.blogspot.fr/2013/03/des-matrices-led-8x8-adafruit-i2c.html

jeudi 7 mars 2013

## Des matrices LED 8x8 AdaFruit (I2C) disponibles chez MCHobby

Que pourrait-il y avoir de mieux qu'une seul LED? Plein de LEDs! Elle permettent de réaliser des petits affichages en utilisant de matrices 8x8 ou des afficheurs 4 chiffres à 7 segments. De telles matrices utilisent le 'multiplexage'. Pour controller 7 segments de leds, il faut donc 14 broches. Cela représente donc beaucoup de broches, il existe bien des [circuits de contrôle comme le MAX7219](https://www.adafruit.com/products/453) pour gérer des matrices mais cela nécessite beaucoup de raccordements qui utilisent une tonne de place.

AdaFruit à bien entendu trouvé une solution à cette pesante situation et a mis au point un breakout permettant de controler des tonnes de LEDs en matrice ou segment sans devoir passer la moitié de la journée à faire des raccordements :-)

Ce breakout existe en deux versions: l'une permettant de controller une mini matrice 8x8. Ajouter des afficheurs LED a vos projets devient totalement trivial.

|  |
| --- |
| [http://4.bp.blogspot.com/-BEf0IZ0sYxI/UTkKfEGNFBI/AAAAAAAADIM/VgmUVGpEXGc/s320/8x8-MINIMAT-GRE.jpg](http://4.bp.blogspot.com/-BEf0IZ0sYxI/UTkKfEGNFBI/AAAAAAAADIM/VgmUVGpEXGc/s1600/8x8-MINIMAT-GRE.jpg) |
| [Mini Matrice 8x8 Verte - I2C - Disponible chez MCHobby](http://mchobby.be/PrestaShop/product.php?id_product=231) |

|  |
| --- |
| [http://3.bp.blogspot.com/-FL5P2Azbyhk/UTkKmPYeH3I/AAAAAAAADIU/BZPOBO0_6kk/s320/8x8-MINIMAT-RED.jpg](http://3.bp.blogspot.com/-FL5P2Azbyhk/UTkKmPYeH3I/AAAAAAAADIU/BZPOBO0_6kk/s1600/8x8-MINIMAT-RED.jpg) |
| [Mini Matrice 8x8 Rouge - I2C - Disponible chez MCHobby](http://mchobby.be/PrestaShop/product.php?id_product=233) |

|  |
| --- |
| [http://2.bp.blogspot.com/-jKo-0y0BHlo/UTkKuy1EzVI/AAAAAAAADIg/WnQeWVWy1XQ/s320/8x8-MINIMAT--YEL.jpg](http://2.bp.blogspot.com/-jKo-0y0BHlo/UTkKuy1EzVI/AAAAAAAADIg/WnQeWVWy1XQ/s1600/8x8-MINIMAT--YEL.jpg) |
| [Mini Matrice 8x8 Jaune - I2C - Disponible chez MCHobby](http://mchobby.be/PrestaShop/product.php?id_product=232) |

Le breakout + afficheur que nous proposons utilise un circuit de contrôle qui prend en charge la partie ardue du travaille. Il dispose d'une horloge interne qu'il utilise pour les opérations de multiplexage. Il utilise également un dispositif d'alimentation à courant constant, idéal pour la gestion des dispositif ultra-brillant mais permet également de préserver la cohérence de la couleur.

La brillance est réglable par pas de 1/16 de la puissance, le tout via une simple interface I2C. Le backpack (carte de contrôle) est livrée avec des connecteur d'adressages (jumper à souder), il est donc possible de connecter jusque 4 matrice 8x8 ou  8 afficheurs 7 segments (ou une combinaison tel que 2 matrices 8x8 et 4 afficheur 7 segment, etc) sur un seul bus I2C.

Tutoriel & Exemple  
Nous disposons également d'un tutoriel d'[utilisation complet sur notre wiki](http://mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK).   
  
Parmi les tutoriels d'AdaFruit, il y a une gestion d'animation à mettant en oeuvre plusieurs de ces produits.  
  
Si ce projet vous intéresse, [vous trouverez la source AdaFruit ici](http://learn.adafruit.com/animating-multiple-led-backpacks).  
  
Plus d'information  
Plus d'informations sur nos fiches produits.

* [Mini Matrice 8x8 - I2C - Verte](http://mchobby.be/PrestaShop/product.php?id_product=231)
* [Mini Matrice 8x8 - I2C - Rouge](http://mchobby.be/PrestaShop/product.php?id_product=233)
* [Mini Matrice 8x8 - I2C - Jaune](http://mchobby.be/PrestaShop/product.php?id_product=232)

# http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK

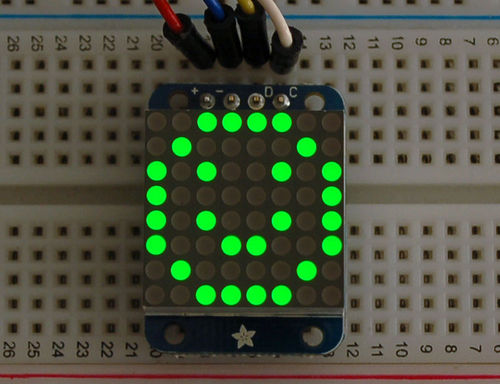
ADF-LED-BACKPACK

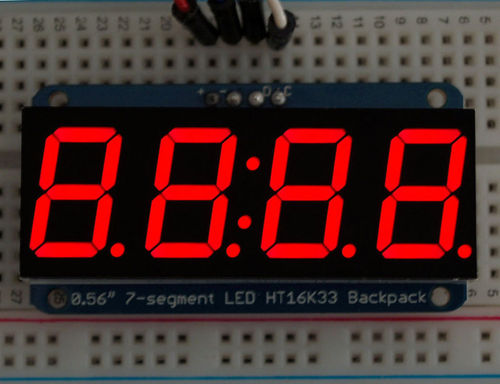
* **Introduction**
* [Matrice 8x8 30mm](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-8x8-Matrice-30)
* [Matrice 8x8 20mm (Mini)](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-8x8-Matrice-20)
* [Aff. 7 Segment 14mm (backpack)](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-7-Segments)
* [Matrice 8x8 Bicolore](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-8x8-Bicolor)
* [Connecter Plusieurs Backpack](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-Backpack-Multiple)
* [Changer adresse I2C](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-I2C-Adresse)
* [F.A.Q.](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-FAQ)
* [Téléchargements](http://www.mchobby.be/wiki/index.php?title=ADF-LED-BACKPACK-Download)
* Où achetez
  + [Afficheurs 4 Chiffres 7 Segments](http://shop.mchobby.be/category.php?id_category=18)
  + [Matrice 8x8 BiColore](http://shop.mchobby.be/product.php?id_product=212)
  + [Matrice 8x8](http://shop.mchobby.be/search.php?orderby=position&orderway=desc&search_query=8x8&submit_search=Rechercher)
  + [Catégorie LEDs](http://shop.mchobby.be/category.php?id_category=18)
  + [breadboard demi-taille](http://shop.mchobby.be/product.php?id_product=53)
  + [Fils pour breadboard](http://shop.mchobby.be/product.php?id_product=34)

Que pourrait-il y avoir de mieux qu'une seul LED? Plein de LEDs! Elle permettent de réaliser des petits affichages en utilisant de matrices 8x8 ou des [afficheurs 4 chiffres à 7 segments](http://shop.mchobby.be/search.php?orderby=position&orderway=desc&search_query=afficheur&submit_search=Rechercher). De telles matrices utilisent le 'multiplexage'. Pour contrôler 7 segments de leds, il faut donc 14 broches. Cela représente donc beaucoup de broches, il existe bien des [circuits de contrôle comme le MAX7219](https://www.adafruit.com/products/453) pour gérer des matrices mais cela nécessite beaucoup de raccordements qui utilisent une tonne de place.

AdaFruit à bien entendu trouvé une solution à cette pesante situation et a mis au point un breakout permettant de contrôler des tonnes de LEDs en matrice ou segment sans devoir passer la moitié de la journée à faire des raccordements :-)

Ce breakout existe en deux versions: l'une permettant de contrôler une mini [matrice 8x8](http://shop.mchobby.be/search.php?orderby=position&orderway=desc&search_query=8x8&submit_search=Rechercher) et des sections de [4 chiffres à 7 segments](http://shop.mchobby.be/product.php?id_product=208) (en [bleu](http://shop.mchobby.be/product.php?id_product=211), [vert](http://shop.mchobby.be/product.php?id_product=209), [jaune](http://shop.mchobby.be/product.php?id_product=210)). Ajouter des afficheurs LED a vos projets devient totalement trivial.

[](http://www.mchobby.be/wiki/index.php?title=Fichier:ADF-LED-BACKPACK-Intro-01.jpg)  
*Crédit: AdaFruit Industries* [*www.adafruit.com*](http://www.adafruit.com)

[](http://www.mchobby.be/wiki/index.php?title=Fichier:ADF-LED-BACKPACK-Intro-02.jpg)  
*Crédit: AdaFruit Industries* [*www.adafruit.com*](http://www.adafruit.com)

Les matrices utilisent un composant qui effectue la partie la plus difficile du travail. Ce composant dispose d'une horloge interne qui lui permet de multiplexer l'affichage. Il utilise un procédé d'alimentation à courant constant pour obtenir une consistance de couleur homogène même avec des LEDs ultra-brillante (NB: la photographie présente l'afficheur en utilisant sont éclairage minimum pour éviter de saturer le capteur CCD de l'appareil photo!). Il peut aussi contrôler la luminosité par 1/16 de d'intensité, le tout par l'intermédiaire d'une simple interface I2C. Le backpacks dispose de cavalier permettant la sélection d'adresse, ce qui permet de connecter jusque 4 matrices 8x8 ou 8 afficheurs 7 segments sur un simple bus I2C.

Le produit est livré en kit composé d'un backpack (assemblé et testé), d'un pinHeader de 4-pin et d'une matrice. Quelques opérations de soudures sont nécessaires pour monter la matrice sur le backpack... mais c'est très facile et ne devrait pas prendre plus de 5 minutes.

Source: [Control small led matrice with ease](http://learn.adafruit.com/adafruit-led-backpack/overview) créé par [LadyAda](http://learn.adafruit.com/users/3) pour AdaFruit Industries. Crédit [AdaFruit Industries](http://www.adafruit.com)

# http://www.sonelec-musique.com/logiciels\_pic\_bases\_mp\_i2c\_pcf8574.html

Dernière mise à jour : **08/09/2013**

#### Présentation

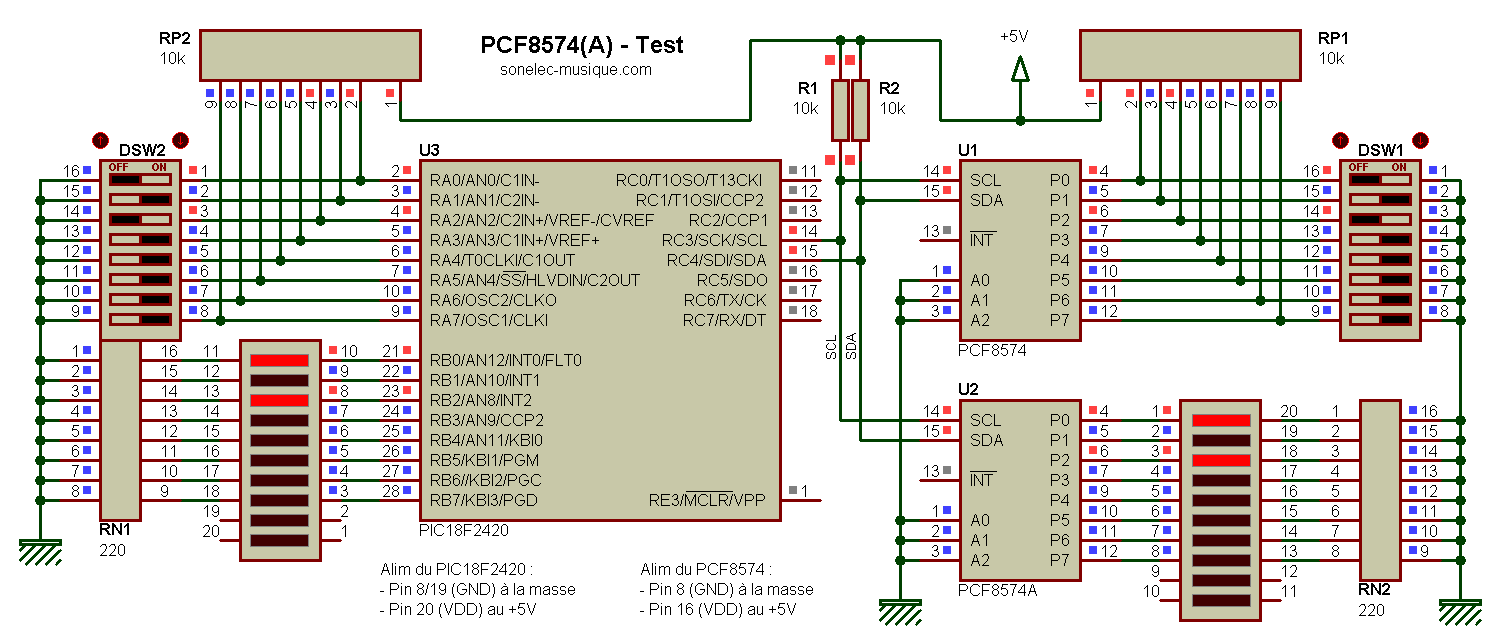
Les exemples décrits ici montrent comment utiliser les expandeurs de port PCF8574 et PCF8574A câblés sur un même bus I2C, en association avec un microcontrôleur de type PIC 18F2420 utilisé en maître. Une partie du texte explique comment adresser ces composants en fonction de l'adresse affectée localement, au niveau physique des PCF8574(A) eux-mêmes.

#### Electronique mise en oeuvre

Les exemples de code logiciel qui suivent montrent comment lire ou écrire une valeur tenant dans un octet (valeur comprise entre 0 et 255), depuis ou vers un PCF8574 grâce à une [liaison bifilaire I2C](http://www.sonelec-musique.com/logiciels_pic_bases_mp_i2c.html).

##### Schéma électronique (001a) pour lecture et écriture

Pour la mise en pratique de ces exemples, le montage dont le schéma suit est mis en oeuvre.

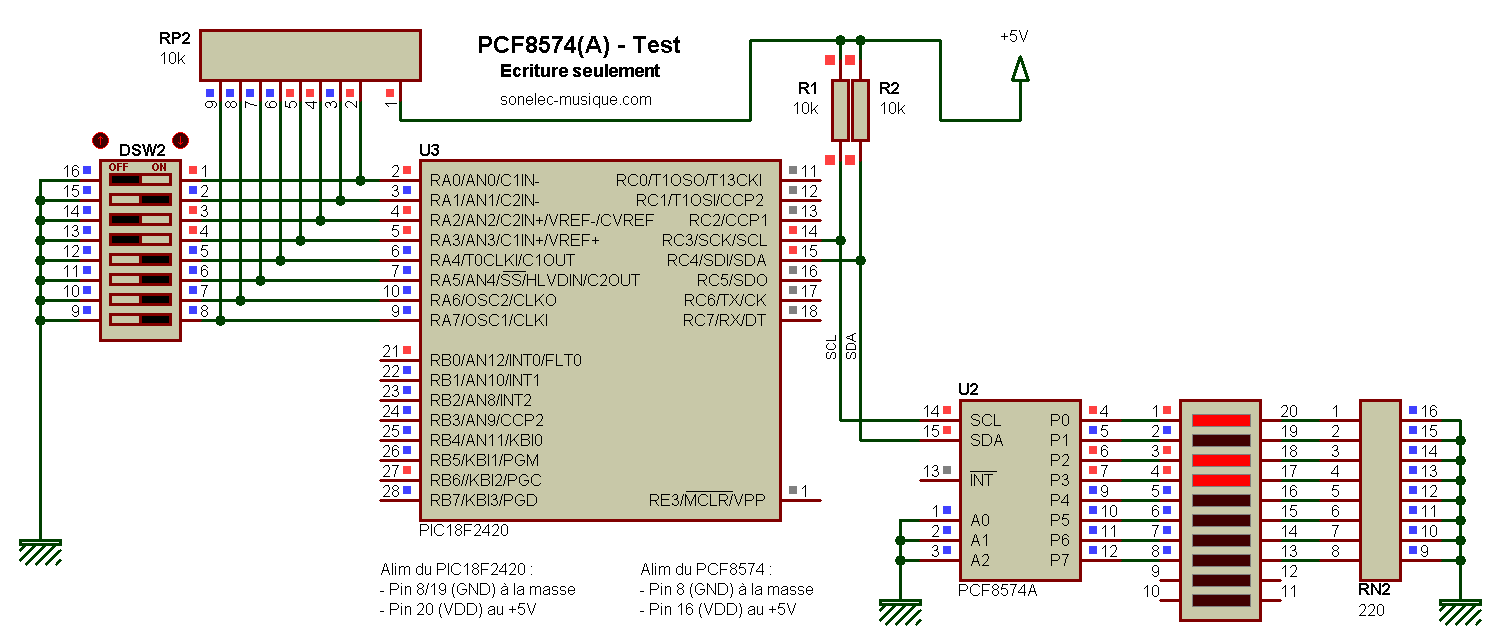
[](http://www.sonelec-musique.com/images3/electronique_pic_tuto_base_i2c_pcf8574_001a.gif)Ce schéma peut sembler un peu complexe et touffu, mais il permet de lire et d'écrire sur le même bus I2C. Et si on y regarde bien, il n'y a finalement pas grand chose : 3 circuits intégrés, quelques interrupteurs, des leds et des réseaux de résistances. En réalité et comme on peut s'en douter un peu, le PIC est le seul composant indispensable pour l'ensemble des opérations de lecture et d'écriture. Les deux schémas suivant montrent les composants réellement utilisés si on ne fait que lire (schéma 001b) ou que écrire (schéma 001c) les données au travres des PCF8574 via le bus I2C.

##### Schéma électronique (001b) pour lecture seulement

##### Cette partie de schéma permet de lire des données provenant du PCF8574 (U1), de telle sorte que ses lignes d'entrées / sorties se comportent en entrées.  [pic_tuto_base_i2c_pcf8574_001b](http://www.sonelec-musique.com/images3/electronique_pic_tuto_base_i2c_pcf8574_001b.gif)

L'état logique des huit lignes du PCF8574 configurée en entrées sont répliquées sur les lignes configurées en sorties du PORTB du PIC. L'état logique de chaque ligne d'entrée du PCF8574 est modifiable grâce aux interrupteurs regroupés dans le bloc DSW1.

##### Schéma électronique (001c) pour écriture seulement

Cette partie de schéma permet d'écrire des données en direction du PCF8574A (U2), de telle sorte que ses lignes d'entrées / sorties se comportent en sorties.   
  
[](http://www.sonelec-musique.com/images3/electronique_pic_tuto_base_i2c_pcf8574_001c.gif)

L'état que prennent les lignes du PCF8574A configurées en sortie dépend de l'état logique des lignes configurées en entrées du PORTA du PIC. L'état logique de chaque ligne d'entrée du port A du PIC est modifiable grâce aux interrupteurs regroupés dans le bloc DSW2.

##### Principe de fonctionnement général

Un PCF8574 et un PCF8574A sont mis en parallèle sur le même bus I2C (voir schéma complet 001a). Ces deux composants fonctionnent exactement de la même façon, la différence réside dans l'adresse de base avec lesquelles on doit les adresser, qui est différente entre les deux. Pour le PCF8574, l'adresse de base est $40, et pour le PCF8574A l'adresse de base est $70. Cette histoire d'adresse ne doit pas vous faire peur, il s'agit juste d'un moyen qui permet à chaque membre d'un réseau I2C de savoir si oui ou non on s'adresse à lui. C'est une façon de faire le tri dans toutes les données qui peuvent circuler sur le bus I2C et qui ne concernent pas forcement tout le monde. Dans le cas présent il n'y a que deux composants reliés sur le même bus I2C, mais on peut aller beaucoup plus loin ! Les PCF8574 et PCF8574A disposent tous deux de trois entrées d'adresse complémentaires, qui permettent de décaller leur adresse d'affectation de base. Ces trois entrées d'adresse complémentaires permettent huit combinaisons possibles, ce qui correspond à huit valeurs d'adresses différentes. Comme les adresses de base des PCF8574 et PCF8574A sont différentes, on peut mettre en parallèle 8 circuits PCF8574 et 8 circuits PCF8574A - chacun avec une adresse différente, ce qui permet de travailler avec 128 lignes de données distinctes (16 x 8 bits). Pour plus de détails, voir paragraphe "Adresses du PCF8574(A)" après le descriptif du code logiciel.

##### Remarque

L'extension du nombre de lignes de données peut egalement être réalisée avec d'autres types de circuits non I2C. Si on veut juste ajouter des lignes de sortie additionnelles, on peut par exemple utiliser le CD4094 qui dispose de 8 sorties (exemples [Voltmètre 005](http://www.sonelec-musique.com/electronique_realisations_voltmetre_005.html) et [Horloge 002](http://www.sonelec-musique.com/electronique_realisations_horloge_002.html)). Et pour un usage en entrée ou sortie, on peut aussi utiliser des multiplexeurs, à condition toutefois de disposer de plus de lignes pour la commande de ces circuits (3 lignes ou plus selon le cas, exemple [Testeur de câble 001](http://www.sonelec-musique.com/electronique_realisations_testeur_cables_001.html)). Si le nombre de lignes disponibles sur le PIC n'est que de deux, alors les expandeurs de port PCF8574 (ou PCF8574A) constituent une solution vraiment pratique et simple à mettre en oeuvre.

#### Lecture et écriture de données avec le PCF8574

Le code exemple qui suit comporte deux ensembles de routines :  
- routines de lecture depuis les lignes d'extension du PCF8574  
- routines d'écriture vers les lignes d'extension du PCF8574**A**  
Pas d'erreur dans les références des circuits utilisés, on met bien en oeuvre deux circuits électroniques PCF8574 différents :  
- un PCF8574 pour utilisation en entrées additionnelles  
- un PCF8574**A** pour utilisation en sorties additionnelles  
Ce choix d'utilisation est arbitraire, on pourrait parfaitement faire l'inverse, c'est-à-dire utiliser le PCF8574 pour disposer de sorties additionnelles et utiliser le PCF8574A pour disposer d'entrées additionnelles.

##### Logiciel

Le code ci-dessous correspond au programme complet.

program electronique\_pic\_tuto\_base\_i2c\_pcf8574\_001a;  
  
const  
 cPCF8574\_W = $40; // -> $4E;  
 cPCF8574\_R = $41; // -> $4F;  
 cPCF8574A\_W = $70; // -> $7E;  
 cPCF8574A\_R = $71; // -> $7F;  
  
var  
 Values: byte;  
   
procedure Main\_Init;  
begin  
 CMCON := $07; // turn off comparators  
 ADCON1 := ADCON1 or $0F; // turn off analog inputs  
 TRISA := $FF; // inputs  
 TRISB := $00; // outputs  
 I2C1\_Init(100000); // I2C comm init  
end;  
  
procedure PCF8574\_Write;  
begin  
 I2C1\_Start; // signal START  
 I2C1\_Wr(cPCF8574A\_W); // addresse PCF8574A  
 Values := PORTA;  
 I2C1\_Wr(Values); // écriture données  
 I2C1\_Stop; // signal STOP  
end;  
  
procedure PCF8574\_Read;  
begin  
 I2C1\_Start; // signal START  
 I2C1\_Wr(cPCF8574\_R); // addresse PCF8574  
 Values := I2C1\_Rd(0); // lecture données  
  PORTB := Values;

I2C1\_Stop; // signal STOP

end;  
  
// programme principal

begin  
 Main\_Init;  
 delay\_ms(500);  
 while true do  
 begin  
 // lecture lignes PCF8574  
 PCF8574\_Read;  
 Delay\_ms(250);  
 // écriture sur lignes PCF8574A  
 PCF8574\_Write;  
 Delay\_ms(250);  
 end;  
end.

##### Descriptif du logiciel

**Procédure Init**Cette routine d'initialisation générale, qui est appelée une seule fois au moment de la mise sous tension du PIC, permet de définir l'orientation des broches des ports A et B, grâce aux lignes TRISA et TRISB. La ligne CMCON = 7 permet de désactiver le fonctionnement des comparateurs du PIC. La ligne ADCON1 permet de désactiver les fonctions liées au convertisseur Analogique / Numérique, toutes les lignes sont en effet utilisées ici de façon logique. La ligne I2C1\_Init permet d'initialiser le module MSSP utilisé pour les fonctions de lecture / écriture sur bus I2C.  
  
**Lecture de données en provenance du PCF8574 (lignes d'entrée additionnelles)**Les données présentes sur les lignes P0 à P7 du PCF8574 (U1) sont répliquées sur le port B du PIC 18F2420 (U3). Si l'entrée P0 du PCF8574 (U1) est à l'état logique haut, alors la led connectée sur la broche RB0 du PIC s'allume. Si l'entrée P1 du PCF8574 (U1) est à l'état logique haut, c'est la led connectée sur la broche RB1 du PIC s'allume, etc.  
  
**Ecriture de données vers le PCF8574 (lignes de sortie additionnelles)**Les données du port A du PIC 18F2420 (U3) sont répliquées sur les lignes du PCF8574A (U2). Si la broche RA0 du PIC est à l'état logique haut, alors la led connectée sur la broche P0 du PCF8574A (U2) s'allume. Si la broche RA1 du PIC est à l'état logique haut, alors c'est la led connectée sur la broche P1 du PCF8574A (U2) qui s'allume, etc.  
  
**Programme principal**Dans le programme principal se déroule des actions qui se répètent indéfiniment, grace à l'emploi d'une boucle infinie (qui ne s'arrête jamais ca aucune condition n'impose son arrêt). Voici précisement ce qui est fait de façon répétée :  
- on lit l'état logique des entrées additionnelles mises à disposition par le PCF8574  
- on attend 250 ms sans rien faire  
- on écrit sur les sorties additionnelles mises à disposition par le PCF8574A  
- on attend 250 ms sans rien faire

#### Adresses des PCF8574 et PCF8574A

Chaque composant I2C esclave possède une adresse qui lui est propre, et la portion de code suivante (extraite du code complet vu ci-avant) précise à quelle numéro de rue on doit frapper pour dialoguer avec les PCF8574(A).

const  
 cPCF8574\_W = $40; // -> $4E;  
 cPCF8574\_R = $41; // -> $4F;  
 cPCF8574A\_W = $70; // -> $7E;  
 cPCF8574A\_R = $71; // -> $7F;

Dans le cas présent, la définition des adresses est faite "en dur" dans le code logiciel via des constantes, avec des valeurs qui collent avec le schéma proposé en début d'article. Mais dans certains cas vous pouvez être ammené à modifier l'adresse de ces composants, ne serait-ce que si vous en mettez plusieurs en parallèle, par exemple huit PCF8574 et huit PCF8574A pour disposer de 128 lignes d'entrées / sorties. Dans ce cas, les numéros d'adresses utilisées dans le logiciel du PIC doivent être adaptées en fonction du composant avec lequel dialoguer. Le procédé d'adressage est assez simple à comprendre : les PCF8574 et PCF8574A disposent d'une adresse fixe (qu'on appelle adresse de base) et cette adresse fixe peut être décalée d'une certaine valeur grâce aux broches d'adresse A0 à A2. Si vous branchez en même temps plusieurs composants sur le même bus, alors chacun doit avoir une adresse différentes parmi les huit possibles. Pour dialoguer avec un PCF8574 configuré avec les broches A2..A0 à 000, vous devez utiliser les adresses $40 et $41 dans le code. Pour dialoguer avec un PCF8574A configuré avec les broches A2..A0 à 110, alors vous devez utiliser les adresses $7C et $7D dans le code.Vous trouverez dans le tableau qui suit les adresses à utiliser selon l'état logique donné aux broches A0 à A2, pour les PCF8574 et PCF8574A.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **A2** | **A1** | **A0** | **PCF8574 Ecriture** | **PCF8574 Lecture** | **PCF8574A Ecriture** | **PCF8574A Lecture** |
| 0 | 0 | 0 | $40 (064d) | $41 (065d) | $70 (112d) | $71 (113d) |
| 0 | 0 | 1 | $42 (066d) | $43 (067d) | $72 (114d) | $73 (115d) |
| 0 | 1 | 0 | $44 (068d) | $45 (069d) | $74 (116d) | $75 (117d) |
| 0 | 1 | 1 | $46 (070d) | $47 (071d) | $76 (118d) | $77 (119d) |
| 1 | 0 | 0 | $48 (072d) | $49 (073d) | $78 (120d) | $79 (121d) |
| 1 | 0 | 1 | $4A (074d) | $4B (075d) | $7A (122d) | $7B (123d) |
| 1 | 1 | 0 | $4C (076d) | $4D (077d) | $7C (124d) | $7D (125d) |
| 1 | 1 | 1 | $4E (078d) | $4F (079d) | $7E (126d) | $7F (127d) |

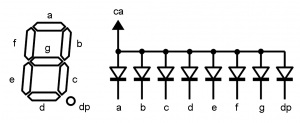
Pour rappel, la valeur effective et complète de l'adresse est codée sur les sept bits de poids fort de l'octet d'adresse envoyé sur le bus I2C, le dernier bit (de poids faible) étant reservé à l'information de direction des données. Le texte qui suit détaille la procédure qui a été suivie pour remplir le tableau précédent. Les valeurs numériques notées selon la forme **xxxd** sont exprimées en décimal et les valeurs notées selon la forme **$xx** sont exprimées en héxadécimal.  
  
**Adresse de base du PCF8574** : **0100 000x**  
- bits 7 à 4 = adresse fixe = **0100** ($40 ou 064d)  
- bits 3 à 1 = adresse variable = **000** ($00 ou 000d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Pour obtenir la valeur de l'adresse complète (part fixe plus part variable), il suffit d'additionner les valeurs de ces trois informations :  
- pour écriture : 064d + 000d + 0 = 064d = $40  
- pour lecture : 064d + 000d + 1 = 065d = $41  
  
**Exemple avec adresse variable du PCF8574 = 001** (A2 = 0, A1 = 0 et A0 = 1), alors on a :  
- bits 7 à 4 = adresse fixe = **0100** ($40 ou 064d)  
- bits 3 à 1 = adresse variable = **001** ($02 ou 002d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Ce qui donne les valeurs suivantes pour l'adresse complète :  
- pour écriture : 064d + 002d + 0 = 066d = $42  
- pour lecture : 064d + 002d + 1 = 067d = $43  
  
**Exemple avec adresse variable du PCF8574 = 111** (A2 = 1, A1 = 1 et A0 = 1), alors on a :  
- bits 7 à 4 = adresse fixe = **0100** ($40 ou 064d)  
- bits 3 à 1 = adresse variable = **111** ($0E ou 014d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Ce qui donne les valeurs suivantes pour l'adresse complète :  
- pour écriture : 064d + 014d + 0 = 078d = $4E  
- pour lecture : 064d + 014d + 1 = 079d = $4F  
  
Notez bien le décallage de 1 bit de la valeur d'adresse variable, idéal pour se planter dans les calculs si on ne fait pas gaffe ! Et bien sûr, même principe pour le PCF8574A.  
  
**Adresse de base du PCF8574A** : **0111 000x**  
- bits 7 à 4 = adresse fixe = **0111** ($70 ou 112d)  
- bits 3 à 1 = adresse variable = **000** ($00 ou 000d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Pour obtenir la valeur de l'adresse complète (part fixe plus part variable), il suffit d'additionner les valeurs de ces trois informations :  
- pour écriture : 112d + 000d + 0 = 112d = $70  
- pour lecture : 112d + 000d + 1 = 113d = $71  
  
**Exemple avec adresse variable du PCF8574A = 011** (A2 = 0, A1 = 1 et A0 = 1), alors on a :  
- bits 7 à 4 = adresse fixe = **0111** ($70 ou 112d)  
- bits 3 à 1 = adresse variable = **011** ($06 ou 006d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Ce qui donne les valeurs suivantes pour l'adresse complète :  
- pour écriture : 112d + 006d + 0 = 118d = $76  
- pour lecture : 112d + 006d + 1 = 119d = $77  
  
**Exemple avec adresse variable du PCF8574A = 110** (A2 = 1, A1 = 1 et A0 = 0), alors on a :  
- bits 7 à 4 = adresse fixe = **0111** ($70 ou 112d)  
- bits 3 à 1 = adresse variable = **110** ($0C ou 012d)  
- bit 0 = bit de direction = **0** pour écriture ou **1** pour lecture  
Ce qui donne les valeurs suivantes pour l'adresse complète :  
- pour écriture : 112d + 012d + 0 = 124d = $7C  
- pour lecture : 112d + 012d + 1 = 125d = $7D  
  
Quand vous utilisez plusieurs PCF8574(A), vous pouvez utiliser les adresses que vous voulez, l'important est que deux composants physiques ne partagent pas la même. Mais vous pouvez adopter la même configuration pour les broches A0..A2 s'il s'agit de PCF8574 et PCF8574A puisque leur adresse de base (partie fixe) est différente. Il n'y a aucune obligation d'utiliser des adresses contigues, ni de commencer avec la part varable de l'adresse à 000. De même, la lecture des différents PCF8574(A) peut se faire dans l'ordre que vous voulez. Vous pouvez ainsi câbler quatre PCF8574(A) sur le même bus et utiliser les adresse suivantes :  
- PCF8574 n°1 : adresses $44 (écriture) et $45 (lecture), broches A2..A0 = 010  
- PCF8574 n°2 : adresses $4E (écriture) et $4F (lecture), broches A2..A0 = 111  
- PCF8574A n°1 : adresses $7A (écriture) et $7B (lecture), broches A2..A0 = 101  
- PCF8574A n°2 : adresses $7E (écriture) et $7F (lecture), broches A2..A0 = 111  
Et lire en premier le PCF8574A n°1, ensuite le PCF8574 n°2, etc.

# <http://home.roboticlab.eu/fr/examples/display/segment_display>

## Afficheur 7 segments à LED

### Théorie

L'indicateur de nombres à 7 segments de LED est un afficheur qui est constitué de 7 LEDs positionnés en forme du chiffre 8. En allumant ou éteignant les LEDs correspondantes, i est possible d'afficher les chiffres de 0 à 9 ainsi qu'un bon nombre de lettres.

[](http://home.roboticlab.eu/_detail/examples/display/segment_display/segment_display_leds.png?id=fr:examples:display:segment_display)

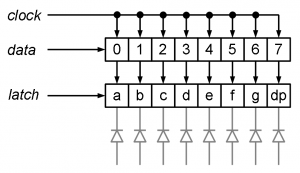
Positionnement des segments de l'indicateur de LED et schéma électrique

Electriquement toutes les anodes des LEDs sont connectées à la broche de l'anode ca. Les LEDs s'allument en inversant la position de leur cathode (a, b, c…). Il existe aussi des connections inversées, où les indicateurs ont une cathode commune cc. Habituellement la plupart des indicateurs de nombres sont utilisés pour afficher des nombres multi numériques - pour ce faire les indicateurs sont équipés d'un segment point dp. En tout et pour tout un indicateur est composé de 8 segments, mais il continu d'être appelé afficheur à 7 segments en accord avec le nombre de segments.

L'afficheur de nombre à LED est très facile à utiliser, il peut être contrôlé directement à partir des broches du micro-contrôleur, mais il existe des drivers spéciaux qui sont capable de contrôler l'afficheur en utilisant moins de broches du micro-contrôleur. Il existe différentes couleurs d'afficheurs qui peuvent être très brillantes et très larges. Enfin, il existe des afficheurs spécifiques composés de segments supplémentaires permettant d'afficher les caractères de l'alphabet Latin.

### Pratique

Il y a un afficheur 7 segments disponible sur la carte du module numérique. Il est contrôlé par l'intermédiaire de l'interface en série A6275. L'interface en série est similaire à SPI, où on utilise les signaux de l'horloge et les signaux de donnée. Contrairement à SPI la fonction chip-select n'est pas utilisée ici, nous la remplacerons par la fonction latch. Les trois voies mentionnées ci-dessus sont connectées à ATmega128 comme suit:

[](http://home.roboticlab.eu/_detail/examples/display/segment_display/segment_display_driver_logic.png?id=fr:examples:display:segment_display)

La construction de l'index du driver de LED avec le segment de l'afficheur correspondant.

* Latch-signal (latch) - PG2
* Clock-signal (clock) - PC7
* Data-signal (data) - PC6

Les données sont transmises en bits en passant par la broche de données. A chaque passage du signal de l'horloge en position haute, le contenu de l'index shift est modifié vers la droite et le bit provenant de la broche de données est lu vers la cellule la plus à gauche. Avec ce 8 bits est transféré vers l'index shift. Si le signal latch est configuré en position haute, la valeur de l'index shift est transférée vers l'index latch, où il y rest jusqu'à un nouveau chargement. Chaque bits de l'index latch est connecté à l'interrupteur de courant de l'un des segments de l'afficheur. Le segment correspond est allumé en position haute du bit-rate.

Pour afficher des nombres sur l'afficheur du HomeLab, la fonctions suivante se trouve dans la librairie du HomeLab.

//

// Set-up of the pins

//

static pin segment\_display\_latch = PIN(G, 2);

static pin segment\_display\_data\_out = PIN(C, 6);

static pin segment\_display\_clock = PIN(C, 7);

//

// Marking card.

// The bits are marking the segments. Lower ranking is A, higher ranking is DP.

//

static const unsigned char segment\_char\_map[11] =

{

0b00111111,

0b00000110,

0b01011011,

0b01001111,

0b01100110,

0b01101101,

0b01111100,

0b00000111,

0b01111111,

0b01100111,

0b01111001 // E like Error

};

//

// Start-up of the 7-segment indicator.

//

void segment\_display\_init(void)

{

// Set latch, data out and clock pins as output

pin\_setup\_output(segment\_display\_latch);

pin\_setup\_output(segment\_display\_data\_out);

pin\_setup\_output(segment\_display\_clock);

}

//

// Displaying number on the 7-segment indicator.

//

void segment\_display\_write(unsigned char digit)

{

unsigned char map;

signed char i;

// Check-up of the number

if (digit > 9)

{

digit = 10;

}

// Number as the card of the segments.

map = segment\_char\_map[digit];

// Latch-signal off

pin\_clear(segment\_display\_latch);

// Sending he bits. Higher ranking goes first.

for (i = 7; i >= 0; i--)

{

// Setting the pin according to the value of the bit of the card.

pin\_set\_to(segment\_display\_data\_out, bit\_is\_set(map, i));

// The clock-signal as high for a moment.

pin\_set(segment\_display\_clock);

\_delay\_us(1);

// The clock-signal as low.

pin\_clear(segment\_display\_clock);

\_delay\_us(1);

}

// Latch-signal on.

pin\_set(segment\_display\_latch);

}

Dans le but d'afficher les chiffres ainsi que la lettre “E”, nous utiliserons une liste constante segment\_char\_map, qui permet d'allumer tous les segments marqués d'un 1 et inversement d'éteindre ceux marqués par un 0. Les bits du plus grand au plus petit (de gauche à droite de la forme binaire) sont dénomé par les segments DP, G, F, E, D, C, B, A. L'interface de contrôle du driver est réalisée a l'aide du logiciel SPI, i.e. en utilisant un logiciel permettant de contrôler les broches de communication dans le programme. Les trois broches sont configurées en sorties avec la fonction segment\_display\_init. Puis la fonction segment\_display\_writepermet l'affichage, qui trouve la carte des segments à partir de la liste et transmet bit à bit toutes les valeurs des segments. Le fréquence de l'horloge avec les temporisation est d'environ 500 kHz.

Le programme suivant est un exemple plus concret d'une utilisation de l'afficheur. Les fonctions utilisées correspondent à celles évoquées ci-dessus. Le programme permet de compter de 0 à 9 avec un intervalle d'environ 1 seconde. Un tel comptage peut être réalisé en prenant en module depuis un nombre plus grand.

//

// The example program of the 7-segment LED indicator of the Home Lab's

// input-output module.

//

#include <homelab/module/segment\_display.h>

#include <homelab/delay.h>

//

// Main program.

//

int main(void)

{

int counter = 0;

// Set-up of the 7-segment indicator.

segment\_display\_init();

// Endless loop.

while (true)

{

// Displaying the ones values of the counter.

segment\_display\_write(counter % 10);

// Counting very long.

counter++;

// Delay for 1 second.

sw\_delay\_ms(1000);

}

}

# <http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html>

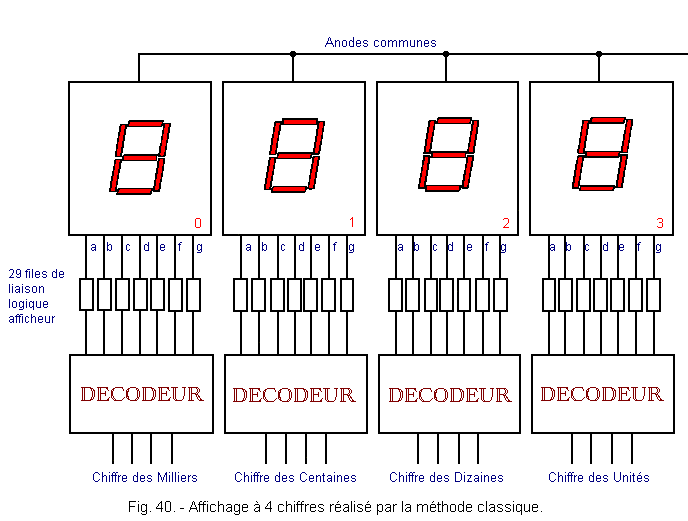
|  |  |  |
| --- | --- | --- |
| **Signets :** | | |
| • [Afficheurs par Matrice LED](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#MATRICE) | • [Autres types d'afficheur](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#AUTRES) | • [Afficheurs à cristaux liquides](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#AFFICHEURS) |
|  | **♦** [Bas de page](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#Page) |  |

**2. 3. - AFFICHEURS MULTIPLEXÉS**

**Supposons que nous voulions afficher un nombre à 4 chiffres. Par la méthode précédente, il nous faudra 4 afficheurs, 4 décodeurs, 28 résistances de liaison soit 29 liaisons sur l'ensemble des afficheurs.**

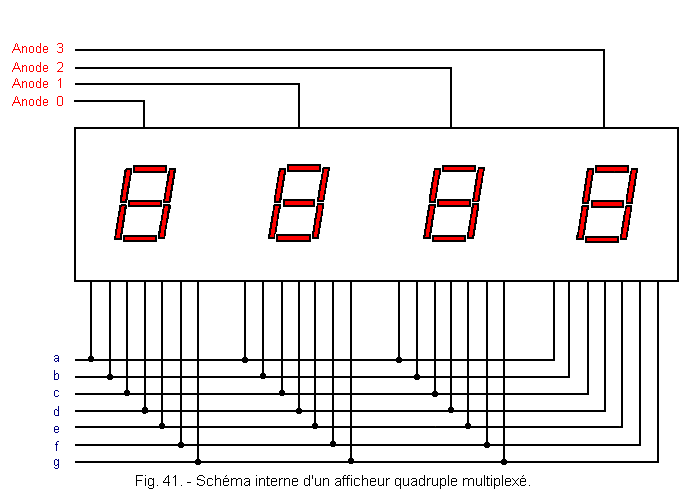
**Ce montage est représenté figure 40.**

**Pour afficher 12 chiffres, ce qui n'est pas rare sur une calculatrice, il faudrait 85 files de liaison et 12 circuits de décodage binaire - 7 segments.**

****

**Afin de simplifier le câblage, il existe des afficheurs à plusieurs chiffres dits multiplexés.**

**La figure 41 représente le schéma interne d'un afficheur multiplexé.**

****

**Le principe de ces afficheurs est très simple :**

**http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/images/astrbul1e.gif  Les diodes LED ont un temps de réponse très court et de toute façon plus court que celui de l'œil humain.**

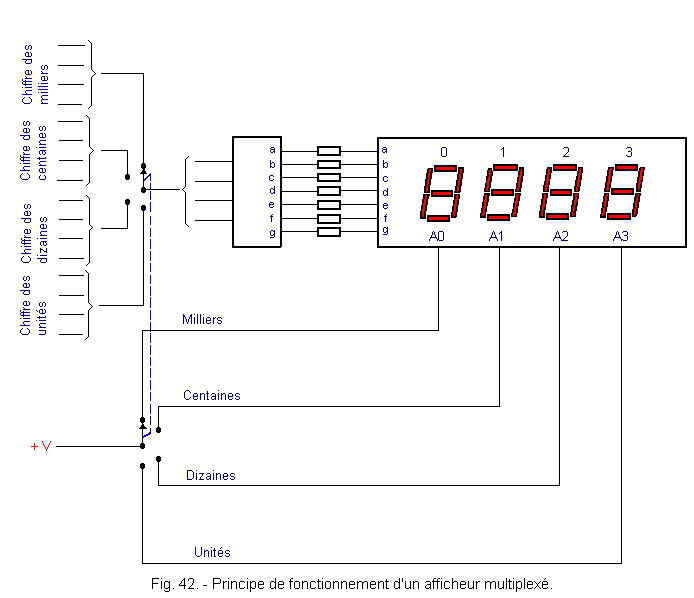
**http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/images/astrbul1e.gif  La persistance rétinienne de l'œil ne permet pas de percevoir le scintillement d'une lampe dont la période de récurrence est inférieure à 20 ms (valeur indicative pour un rapport cyclique de 1).**

**Il suffit donc d'envoyer successivement sur les lignes bus a, b, c, d, e, f, g les codes correspondants aux chiffres à afficher et ce de manière cyclique : unité, dizaine, centaine, millier, unité, dizaine, centaine, millier, unité,.... et ainsi de suite.**

**On appelle lignes bus : un ensemble de conducteurs permettant de distribuer simultanément l'information à plusieurs circuits.**

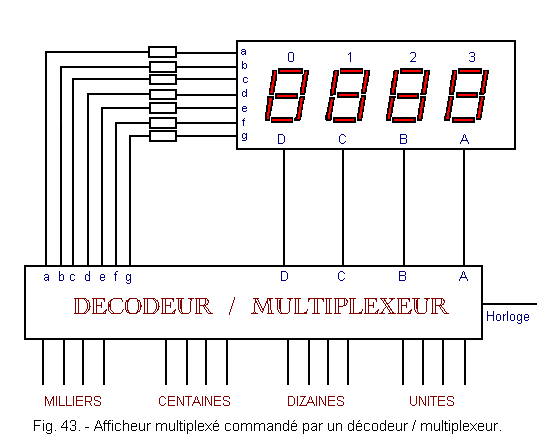
**Il sera bien évidemment nécessaire de sélectionner simultanément l'anode de l'afficheur concerné afin que le chiffre n'apparaisse que sur la décade et seulement la décade concernée : unité, dizaine,....**

**La distribution de signaux est réalisée au moyen d'un commutateur électronique que nous appellerons multiplexeur et qui sera examiné ultérieurement, c'est pourquoi il est schématisé par un contact dans la figure 42.**

****

**Il existe des décodeurs spécialisés pour circuits afficheurs multiplexés.**

**La figure 43 montre l'utilisation d'un tel décodeur.**

****

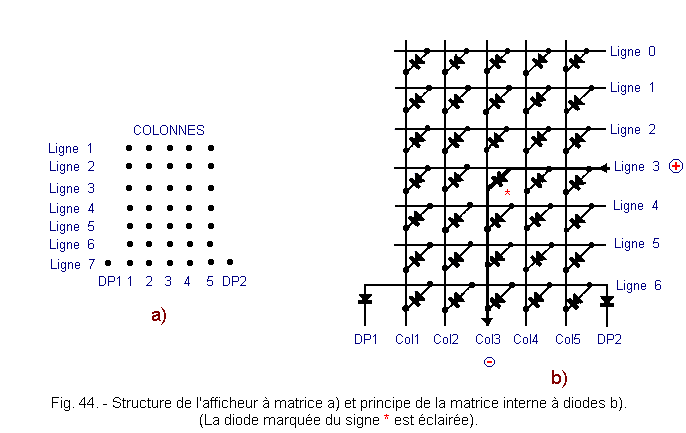
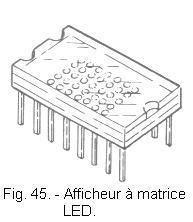
**Dans cette figure est représenté un bloc de 4 afficheurs ainsi que le décodeur / multiplexeur qui lui est associé.**

**On peut rencontrer également des blocs d'afficheurs beaucoup plus importants contenant des points décimaux et des signes (plus ou moins).**

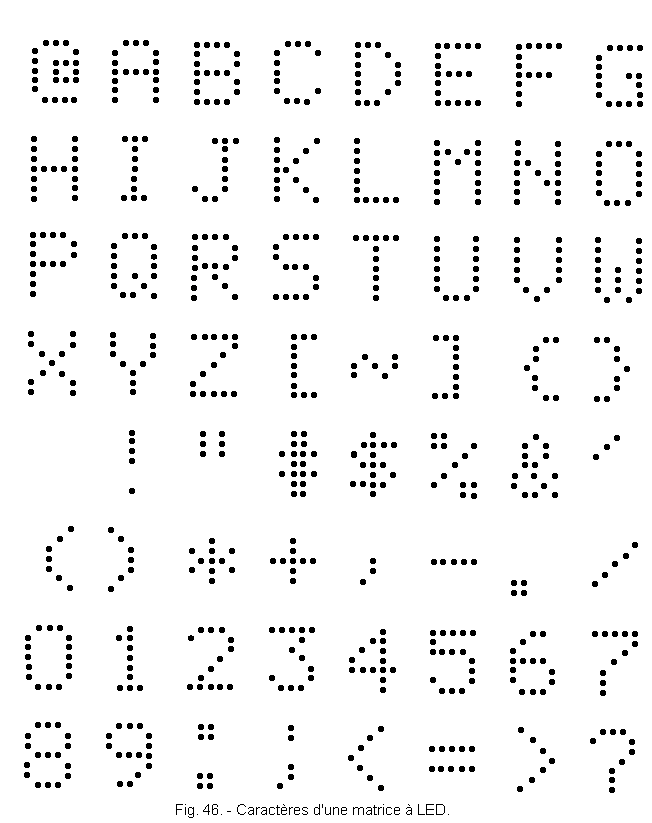
**Dans ce domaine chaque constructeur présente une gamme de produits différents qui peut répondre à tous les besoins.**

**[HAUT DE PAGE](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#HAUT)2. 4. - AFFICHEURS PAR** **MATRICE LED**

**Un type d'afficheur très utilisé emploie 37 LED en forme de points disposés selon une matrice à 7 lignes et 5 colonnes plus 2 points décimaux (DP1 et DP2), l'un situé à droite et l'autre à gauche (figures 44 et 45).**

**[](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#Ces)**

**Comme pour l'afficheur à 7 segments en allumant les LED appropriées, il est possible d'obtenir les 10 chiffres décimaux ; de plus, grâce au nombre de LED plus important, il est possible de représenter d'autres caractères parmi lesquels toutes les lettres de l'alphabet et différents signes comme +, -, /, (,) et d'autres encore, comme représenté à la figure 46.**

****

**Ces afficheurs nécessitent du fait du plus grand nombre de LED des circuits de décodage plus complexes.**

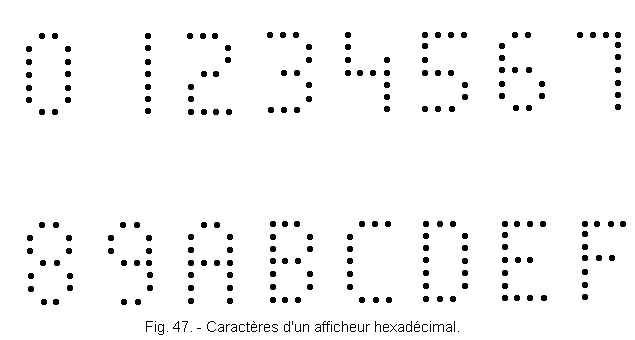
**En ce qui concerne la commande de ces circuits, il est facile de comprendre que vu le principe de la matrice à diode (**[**figure 44**](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#Figure44)**) si l'on veut par exemple éclairer les diodes aux intersections ligne 3 colonne 3 et ligne 4 colonne 4, on allumera forcément la diode située ligne 4 colonne 3.**

**On peut donc en déduire qu'il est nécessaire de disposer d'un circuit réalisant le multiplexage des entrées ligne et colonne de telle sorte que l'allumage des diodes ne soit pas simultané mais ait lieu les uns après les autres afin de supprimer ce défaut.**

**Ce type de circuit, appelé également circuit de balayage, peut être externe ou interne à l'afficheur.**

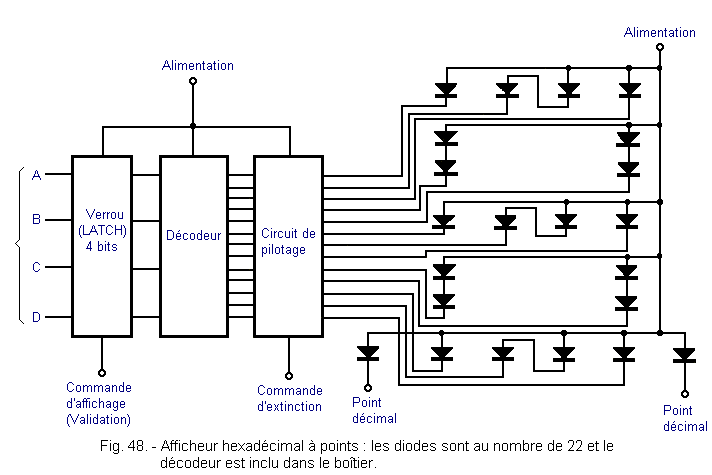
**Les multiplexeurs nous étant encore inconnus, nous n'en dirons pas plus long sur ce type d'afficheur.**

**Il existe également un autre type d'afficheur à points dont l'utilisation est plus simple et dans lesquels les caractères disponibles sont limités à ceux du code hexadécimal autrement dit aux dix chiffres de 0 à 9 et aux lettres de A à F, comme vous pouvez le voir figure 47.**

****

**Les LED sont moins nombreuses, soit 20 pour les caractères et 2 pour le point décimal et ces circuits ne nécessitent donc pas de multiplexage.**

**Dans la figure 48, vous pouvez voir comment sont disposés les diodes LED et les circuits de mémoire (Latch), de décodage et de pilotage, incorporés dans le même boîtier que les LED.**

****

**Le décodeur :**

**La caractéristique la plus importante de ce décodeur est le fait que celui-ci soit inclus dans le boîtier ce qui permet d'envoyer directement le nombre binaire fourni par le compteur sur l'afficheur.**

**Circuit latch à 4 bits :**

**Il sert à emmagasiner et à afficher au moment souhaité le nombre binaire fourni.**

**Cela est très utile car souvent le nombre à visualiser est disponible uniquement pendant un très court instant insuffisant pour être perçu par l'œil. En le mémorisant dans le latch, on peut le visualiser ensuite pendant un temps indépendant de la logique pour qu'il soit vu correctement.**

**Circuit de pilotage :**

**Ce circuit permet de fournir aux LED le courant d'alimentation nécessaire au moyen d'un générateur de courant constant. Il est doté d'une borne sur laquelle on peut appliquer un signal qui commande l'extinction simultanée de toutes les LED. Ceci est très utile lorsque des nombres à plusieurs chiffres doivent être affichés avec un afficheur indépendant pour chaque chiffre.**

**Dans le cas où l'on veut afficher un nombre très inférieur à la capacité maximale de l'afficheur, ce nombre apparaîtra avec un ou plusieurs 0 à gauche.**

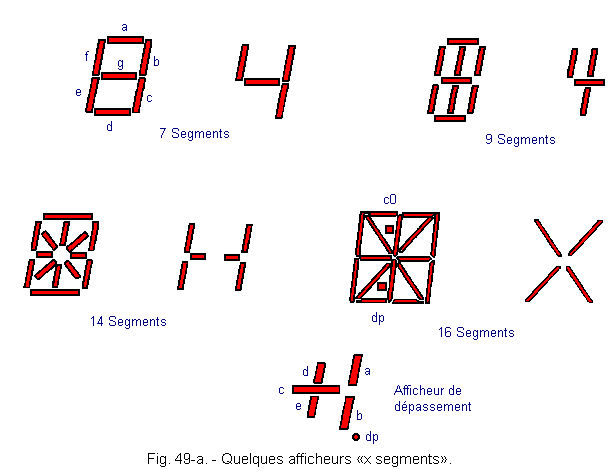
**Par exemple avec 4 afficheurs, le nombre hexadécimal que l'on peut afficher sera FFFF. Un nombre beaucoup plus petit tel 23 apparaîtra sous la forme 0023.**

**Le nombre affiché est parfaitement compréhensible mais l'on peut souhaiter pour des raisons d'esthétique supprimer ces 0. Dans ce cas en agissant sur l'entrée d'extinction, on peut les effacer.**

**[HAUT DE PAGE](http://electronique-et-informatique.fr/Electronique-et-Informatique/Digit/Digit_10TS3.html#HAUT)2. 5. -** **AUTRES TYPES D'AFFICHEURS**

**2. 5. 1. - AUTRES AFFICHEURS À LED**

**Il existe également d'autres types d'afficheurs à LED à 9, 14 ou 16 segments (figure 49-a).**

****

**A chaque fois, il suffira de disposer du décodeur adéquat.**

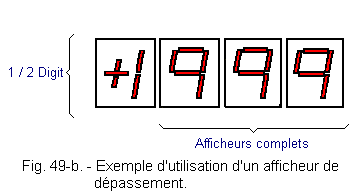
**Le plus évolué (16 segments) permet toutes les représentations alphanumérique.**

**Il existe des afficheurs dits afficheurs à dépassement (appelés parfois demi Digit).**

**Ces afficheurs ne comportent que 5 segments, et servent à indiquer le signe + ou -, ainsi que le chiffre 1.**

**Dans un voltmètre à affichage numérique par exemple, le premier afficheur (le plus à gauche) sera 1 / 2 Digit.**

**La figure 49-b représente un affichage de 3 Digits 1 / 2.**

****

# <http://skyduino.wordpress.com/2012/05/01/arduino-de-electronique-avec-capteur-tilt-et-afficheur-7-segments/>

## Dé électronique avec capteur tilt et afficheur 7 segments

Aujourd’hui je vais vous présenter un projet très simple, tellement simple que je n’ai jamais eu l’idée de faire un article dessus :)

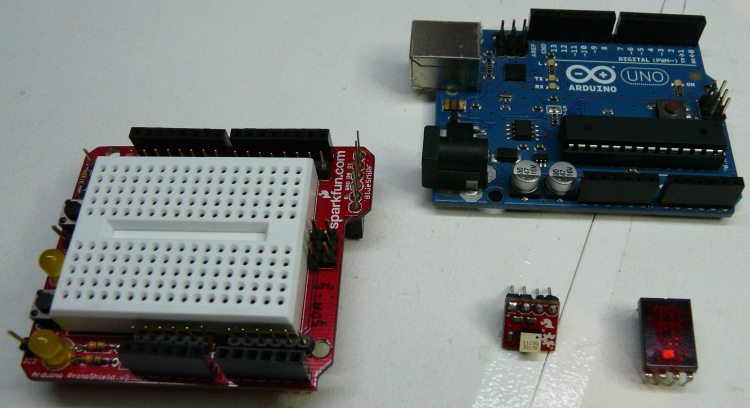
Un des montages classique que l’on réalise quand on débute en électronique c’est le « dé électronique ».  
6 leds, quelques CI logique, et un capteur de vibrations, c’est le montage de base pour amuser les jeunes débutants.  
Ca doit rappeler des souvenirs à ceux ayant déjà monté un kit de la marque velleman (ou autres).

Et si on remettez ce montage au goût du jour ? Avec une petite touche de magie « Arduinesque » ;)

**Le sujet du jour :** fabriquer un dé électronique avec une carte arduino !

–

**La matériel :**

[](http://skyduino.files.wordpress.com/2012/04/p1040672.jpg)

Pour ce projet on va rester simple, mais pas trop !

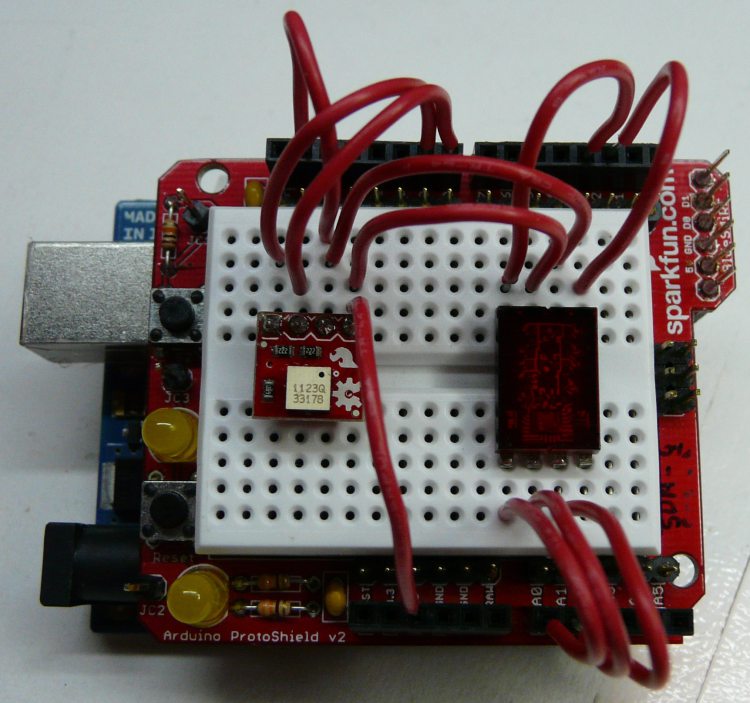
Pour l’affichage pas question d’utiliser 6 leds comme dans la plupart des kit du commerce, non, un afficheur 7 segments c’est beaucoup plus marrant :)  
De même pourquoi utiliser un capteur piezo pour détecter les choc, quand on dispose d’un capteur tilt.

Pourquoi faire simple quand on peut faire compliqué ? hein ;)

Le matériel à proprement parler :  
Une carte arduino,  
Un capteur tilt (j'en est déjà parlé il y a peu dans un autre article),  
Un afficheur 7 segments avec électronique intégré (pareil j'ai fait un article dessus il y a peu de temps),  
Une breadboard pour faciliter le câblage,  
Des fils de câblage

Pour l’afficheur 7 segments, il est aussi tout à fait possible d’utiliser un afficheur 7 segments classique, avec un contrôleur BCD -> 7 segments externes (comme un CD4543 par exemple).

**Le câblage :**

[](http://skyduino.files.wordpress.com/2012/04/p1040665.jpg)

Arduino -> capteur tilt  
VCC -> VCC  
GND -> GND  
D8 -> S2  
D9 -> S1

Arduino -> 7seg  
VCC -> VCC  
GND -> GND  
D3 -> Data 1  
A0 -> Data 2  
A1 -> Data 4  
A2 -> Data 8  
D2 -> LATCH\_STROBE  
(Le pinmapping du 7seg est un peu bizarre pour faciliter mon câblage sur la breadboard)

La broche A3 doit être laissée libre, ou doit juste être reliée à un fils flottant, afin de servir d’antenne pour initialiser le générateur de nombre aléatoire.

**Le code :**

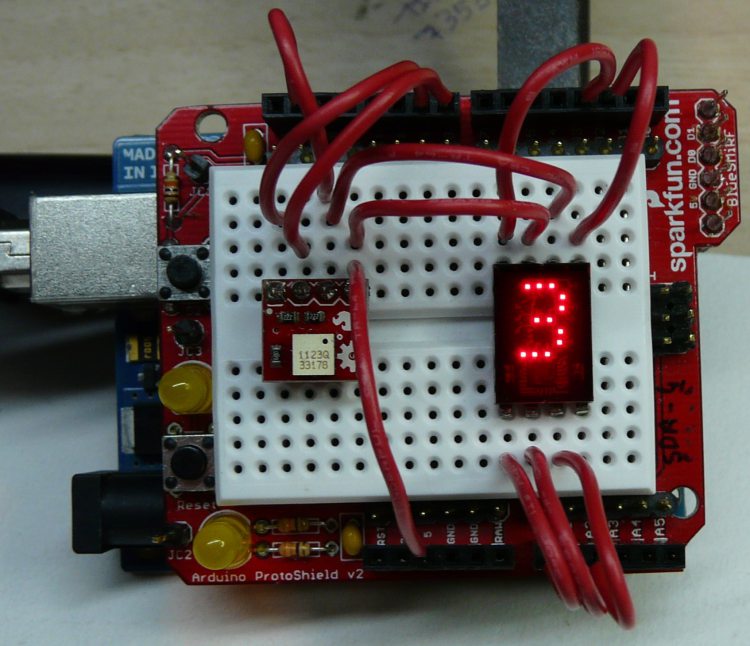
(Rien de bien compliqué, de plus, beaucoup de morceaux de code proviennent d’autre articles du blog)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | /\* Broche analogique libre servant pour RandomSeed() \*/  #define RAND\_SEED\_PIN A3    /\* Broches de contrôle du 7seg \*/  #define LATCH\_STROBE 2  #define LATCH\_DATA\_A 3  #define LATCH\_DATA\_B A0  #define LATCH\_DATA\_C A1  #define LATCH\_DATA\_D A2    /\* Broche du capteur tilt \*/  #define \_S1 9  #define \_S2 8    /\* Fonction qui retourne l'orientation du capteur tilt \*/  byte getPos() {    byte n = digitalRead(\_S1);    n += 2 \* digitalRead(\_S2);    return n;  }    /\* Fonction qui affiche une valeur sur le 7seg \*/  void affichage(uint8\_t valeur) {    valeur &= 0x0F; // Sécurité : 0 < valeur < 15      /\* Envoi des données sur le latch \*/    digitalWrite(LATCH\_DATA\_A, bitRead(valeur, 0));    digitalWrite(LATCH\_DATA\_B, bitRead(valeur, 1));    digitalWrite(LATCH\_DATA\_C, bitRead(valeur, 2));    digitalWrite(LATCH\_DATA\_D, bitRead(valeur, 3));      /\* Affichage des données \*/    delayMicroseconds(100);    digitalWrite(LATCH\_STROBE, LOW);    delayMicroseconds(100);    digitalWrite(LATCH\_STROBE, HIGH);  }    /\* setup() \*/  void setup() {    /\* Mise en sortie des broches de contrôle \*/    pinMode(LATCH\_STROBE, OUTPUT);      /\* Mise en sortie des broches du 7seg \*/    pinMode(LATCH\_DATA\_A, OUTPUT);    pinMode(LATCH\_DATA\_B, OUTPUT);    pinMode(LATCH\_DATA\_C, OUTPUT);    pinMode(LATCH\_DATA\_D, OUTPUT);      /\* Mise en sortie des broches du capteur tilt \*/    pinMode(\_S1, INPUT);    pinMode(\_S2, INPUT);      /\* Initialisation du générateur de nombre aléatoire \*/    randomSeed(analogRead(RAND\_SEED\_PIN));  }    /\* loop() \*/  void loop() {    static byte oldPos = 0; // Ancienne valeur du capteur    byte pos = getPos();    // Valeur actuelle du capteur      /\* Si le capteur a bouger \*/    if(pos != oldPos) {      affichage(random(0, 7)); // Affichage d'un nombre aléatoire entre 0 et 7      delay(1000);             // Délai 1s      oldPos = getPos();       // Sauvegarde du mouvement    }  } |

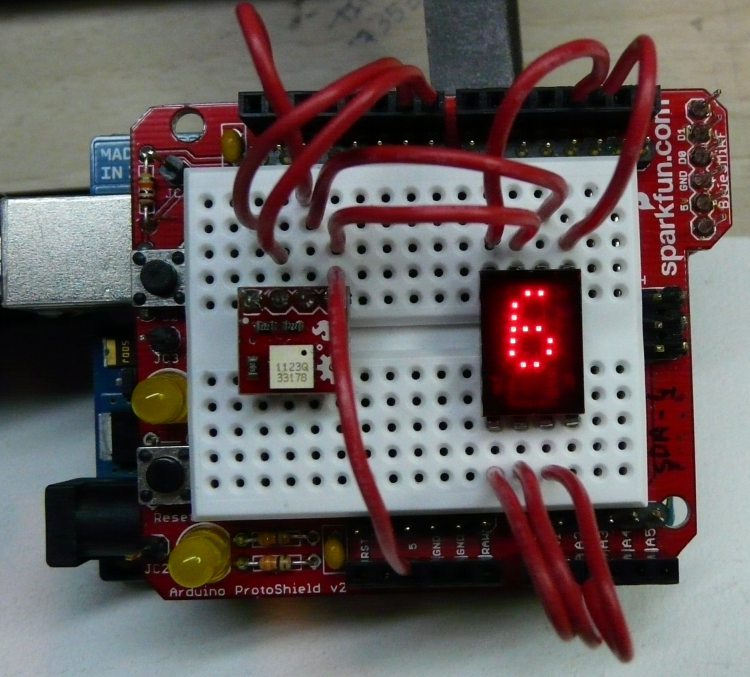
–

**Le résultat :**

\*On secoue comme dans la pub orangina\*

[](http://skyduino.files.wordpress.com/2012/04/p1040670.jpg)

\*Et on re-secoue \*

[](http://skyduino.files.wordpress.com/2012/04/p1040668.jpg)

# Affichages contrôlés par i2c

**5x7 Matrix help**

<https://www.raspberrypi.org/forums/viewtopic.php?t=45698>

[Fri May 31, 2013 11:03 am](https://www.raspberrypi.org/forums/viewtopic.php?p=361026&sid=59d8fbd593014206482e737a470de377#p361026)

Hello all,  
I have run into an issue writing the code to power a 5x7 matrix design in this <http://images.elektroda.net/16_1201732409.jpg> format. Im not sure how i would go about isolating just one LED. Can a gpio pin in its 0 or low position double as ground? If not i cant visualize how this works. Thanks in advance for any help!

[](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=14062&sid=59d8fbd593014206482e737a470de377)

[mahjongg](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=14062&sid=59d8fbd593014206482e737a470de377)

Forum Moderator  
Forum Moderator

**Posts:** [11710](https://www.raspberrypi.org/forums/search.php?author_id=14062&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Sun Mar 11, 2012 12:19 am

**Location:** South Holland, The Netherlands

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361031)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361031&sid=59d8fbd593014206482e737a470de377)

[Fri May 31, 2013 11:12 am](https://www.raspberrypi.org/forums/viewtopic.php?p=361031&sid=59d8fbd593014206482e737a470de377#p361031)

You output a high on only one of the column lines (which need series resistors by the way), and output a low on all others (or use as input, as the row lines below).  
  
On the row lines you output a low on only one row, and program the GPIO as input for all others.  
  
programming a GPIO as input outputs neither a high, nor a low, but turns it off (high impedance).

[turnbullangus](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=65437&sid=59d8fbd593014206482e737a470de377)

**Posts:** [3](https://www.raspberrypi.org/forums/search.php?author_id=65437&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Wed Apr 10, 2013 9:46 pm

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361047)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361047&sid=59d8fbd593014206482e737a470de377)

[Fri May 31, 2013 11:39 am](https://www.raspberrypi.org/forums/viewtopic.php?p=361047&sid=59d8fbd593014206482e737a470de377#p361047)

So if I want to light up just one led I program that leds column as a high output(with all other column being low output) and I program its row as a low output (with the rest of the rows being inputs)?  
  
And where would the resistors need to be in the diagram I linked?

[](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=14062&sid=59d8fbd593014206482e737a470de377)

[mahjongg](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=14062&sid=59d8fbd593014206482e737a470de377)

Forum Moderator  
Forum Moderator

**Posts:** [11710](https://www.raspberrypi.org/forums/search.php?author_id=14062&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Sun Mar 11, 2012 12:19 am

**Location:** South Holland, The Netherlands

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361327)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361327&sid=59d8fbd593014206482e737a470de377)

[Fri May 31, 2013 9:40 pm](https://www.raspberrypi.org/forums/viewtopic.php?p=361327&sid=59d8fbd593014206482e737a470de377#p361327)

doesn't really matter, as long as at least one resistor is in series of every possible LED path.  
because it needs the minimum number of resistors (5) I would put them in series with the column lines.  
A good value would be 120 Ohm, for about 10mA.

[turnbullangus](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=65437&sid=59d8fbd593014206482e737a470de377)

**Posts:** [3](https://www.raspberrypi.org/forums/search.php?author_id=65437&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Wed Apr 10, 2013 9:46 pm

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361539)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361539&sid=59d8fbd593014206482e737a470de377)

[Sat Jun 01, 2013 9:54 am](https://www.raspberrypi.org/forums/viewtopic.php?p=361539&sid=59d8fbd593014206482e737a470de377#p361539)

Okay that makes sense. I've been looking around while trying to write this script and I can't seem to figure out how to change the setup of a pin from input to output mid script and back again. Is this possible?

[tvoverbeek](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=55705&sid=59d8fbd593014206482e737a470de377)

**Posts:** [97](https://www.raspberrypi.org/forums/search.php?author_id=55705&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Mon Feb 04, 2013 9:50 am

**Location:** Fieberbrunn, Austria

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361653)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361653&sid=59d8fbd593014206482e737a470de377)

[Sat Jun 01, 2013 1:47 pm](https://www.raspberrypi.org/forums/viewtopic.php?p=361653&sid=59d8fbd593014206482e737a470de377#p361653)

See the tutorials on <http://w8bh.net> under Raspberry Pi -> Pi matrix.  
This is an 8x8 matrix, but works the same as your 5x7.  
Check also Gordon Henderson's wiringPi library (<http://wiringpi.com>), especially the gpio utility which can be used from scripts (<http://wiringpi.com/the-gpio-utility/>). You can change modes for pins and much more and you do not have to run as root. (gpio is a setuid utility).  
HTH  
  
Ton van Overbeek

[](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=10464&sid=59d8fbd593014206482e737a470de377)

[gordon@drogon.net](https://www.raspberrypi.org/forums/memberlist.php?mode=viewprofile&u=10464&sid=59d8fbd593014206482e737a470de377)

**Posts:** [2016](https://www.raspberrypi.org/forums/search.php?author_id=10464&sr=posts&sid=59d8fbd593014206482e737a470de377)

**Joined:** Tue Feb 07, 2012 2:14 pm

**Location:** Devon, UK

**Contact:** [Website](http://projects.drogon.net/)

### [Re: 5x7 Matrix help](https://www.raspberrypi.org/forums/viewtopic.php?t=45698#p361824)

* [Quote](https://www.raspberrypi.org/forums/posting.php?mode=quote&f=41&p=361824&sid=59d8fbd593014206482e737a470de377)

[Sat Jun 01, 2013 6:50 pm](https://www.raspberrypi.org/forums/viewtopic.php?p=361824&sid=59d8fbd593014206482e737a470de377#p361824)

tvoverbeek wrote:See the tutorials on <http://w8bh.net> under Raspberry Pi -> Pi matrix.  
This is an 8x8 matrix, but works the same as your 5x7.  
Check also Gordon Henderson's wiringPi library (<http://wiringpi.com>), especially the gpio utility which can be used from scripts (<http://wiringpi.com/the-gpio-utility/>). You can change modes for pins and much more and you do not have to run as root. (gpio is a setuid utility).  
HTH  
  
Ton van Overbeek

Also look at this:  
  
[https://projects.drogon.net/piface-abus ... s-outputs/](https://projects.drogon.net/piface-abuse-using-inputs-as-outputs/)  
  
It's not a 5x7 dot display, but a multiplexed 7segment display - the principle of multiplexing is the same - you enable one column (digit in this case), then set the row outputs (segments here), repeat for each column. That one actually enables up to all 7 segments each time as it's a common cathode display and I have higher current drivers on the PiFace.  
  
This also has an 8x8 LED matrix:  
  
<https://projects.drogon.net/the-game-of-life/>  
  
Hm. the video for that seems to have been edited out, but here it is:  
  
<http://www.youtube.com/watch?v=g5cSvVGCX_A>  
  
So there are 2 strategys - the easiest is to use 5 resistors, one on each columns in which case you can only drive one LED at a time. You don't need to move pin from input mode to output, all 12 pins are outputs, and set the columns low and the rows high to start with.  
  
To light up an LED, you set the column high, then set the 7 row pins low, one after the other, retuning each one high after - only light one LED at a time.  
  
then you move on to the next column - the first col pin goes high again, the next goes low, and you repeat the 7 row LED lights one after the other.  
  
Repeat that sequence fast enough and you'll see a solid image...  
  
-Gordon

**--**   
Gordons projects: <https://projects.drogon.net/>

# Beginner’s Guide to the PI MATRIX-Part 1-

<http://w8bh.net/pi/PiMatrix1.pdf>

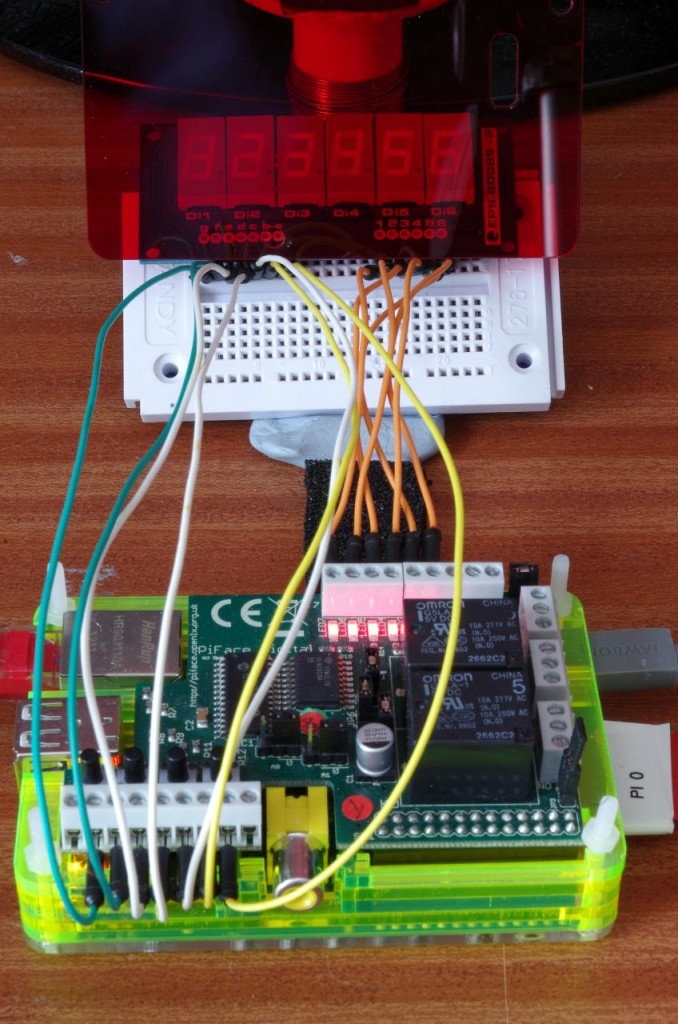
# PiFace abuse: Using Inputs as Outputs

<https://projects.drogon.net/piface-abuse-using-inputs-as-outputs/>

Posted on [May 27, 2013](https://projects.drogon.net/piface-abuse-using-inputs-as-outputs/) by [Gordon](https://projects.drogon.net/author/gordon/)

Did you know that the 8 inputs on the PiFace can be used as outputs too? Well they can, and this is something that I understand the boffins in Manchester who designed it had in-mind too… So in-between barbecues this (UK) bank holiday weekend, I’ve had some fun and adapted something I did on the Gertboard to the PiFace – Driving a 6-digit 7-segment display.

Here it is:

[](https://projects.drogon.net/wp-content/uploads/2013/05/piface-io.jpg)You can just make out the numbers 123456 on the LED digits at the top…

So how does it work: The Piface really is nothing more than an interface board with a standard SPI based GPIO expansion chip – the MCP23S17 which as 16 IO pins. 8 of these are connected to a ULN2803 open-collector buffer (darlington) chip – 8 of those 8 outputs go to on-board relays, the other 8 are uncommitted. On the input side – the 8 inputs pins go via 330Ω resistors to the terminals (and 4 buttons).

Pins 0-7 go to the ULN2803 and to the relays and output terminals and pins 8-15 go to the 330Ω resistors, the buttons and input terminals.

If we access the MCP23S17 directly, rather than use the [***wiringPi***](http://wiringpi.com/) PiFace devLib for it, then we can re-provision the input pins as outputs and use them to drive things like LEDs.

(Well, not quite good enough for my display above – it’s over 25 years old, so old LEDs which have faded, and those old LEDs did need more current to drive them than the 330Ω resistors are allowing, but it’s the principle of the thing here!)

So how do we drive LED digits – easy here. We have 8 inputs which we’re using as outputs, so we can connect 7 to the 7 segments on the display and the common pin on each digit can go to one of the open collector drivers. To light up a single digit, we program the 7 segments as required, and enable the common line for that digit. Current flows out of the input pin, through the LED segments, and via the common pin for that segment into the open-collector driver on the PiFace and to ground.

To light up many digits, we repeat the above for each digit. ie

For each digit:

Set 7-segment pattern

Enable digit common-line

delay a short time

Disable digit common-line

This way, we can cycle through each of the 6 digits in-turn and if we do it fast enough then we don’t notice it happening (Look-up Persistence of vision)

We can illuminate all 7 segments at once, if required (e.g. to display an 8) as each one has it’s own limiting resistor and the “sink” pin goes via a darlington driver that can handle the combined current.

One thing to note here: This will only work with common-cathode type displays.

So there you have it – not quite “abuse” as it’s really by design, but something else to think about when using your PiFace board – if you need a 3.3v output and you have spate inputs, then re-provision one as an output and off you go.

Here’s the code for anyone interested:

/\*

\* 7segments.c:

\* PiFace "abuse". Use the inputs as outputs to drive a 7-segment

\* LED display (common cathode) - connect the segments to the

\* input pins, re-provisioned as outputs and use the darlington

\* outputs as current sinks for each digit.

\*

\* Copyright (c) 2013 Gordon Henderson

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#undef PHOTO\_HACK

#include <wiringPi.h>

#include <mcp23s17.h>

#include <stdio.h>

#include <time.h>

#include <ctype.h>

#include <string.h>

/\*

\* Segment mapping

\*

\* --a--

\* | |

\* f b

\* | |

\* --g--

\* | |

\* e c

\* | |

\* --d-- p

\*/

#define PF 200

// GPIO Pin Mapping

static int digits [6] = { PF+ 7, PF+ 6, PF+ 5, PF+ 4, PF+ 3, PF+ 2 } ;

static int segments [7] = { PF+15, PF+14, PF+13, PF+12, PF+11, PF+10, PF+ 9 } ;

static const int segmentDigits [] =

{

// a b c d e f g Segments

1, 1, 1, 1, 1, 1, 0, // 0

0, 1, 1, 0, 0, 0, 0, // 1

1, 1, 0, 1, 1, 0, 1, // 2

1, 1, 1, 1, 0, 0, 1, // 3

0, 1, 1, 0, 0, 1, 1, // 4

1, 0, 1, 1, 0, 1, 1, // 5

1, 0, 1, 1, 1, 1, 1, // 6

1, 1, 1, 0, 0, 0, 0, // 7

1, 1, 1, 1, 1, 1, 1, // 8

1, 1, 1, 1, 0, 1, 1, // 9

1, 1, 1, 0, 1, 1, 1, // A

0, 0, 1, 1, 1, 1, 1, // b

1, 0, 0, 1, 1, 1, 0, // C

0, 1, 1, 1, 1, 0, 1, // d

1, 0, 0, 1, 1, 1, 1, // E

1, 0, 0, 0, 1, 1, 1, // F

0, 0, 0, 0, 0, 0, 0, // blank

} ;

// display:

// A global variable which is written to by the main program and

// read from by the thread that updates the display. Only the first

// 6 characters are used.

char display [8] ;

/\*

\* displayDigits:

\* This is our thread that's run concurrently with the main program.

\* Essentially sit in a loop, parsing and displaying the data held in

\* the "display" global.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

PI\_THREAD (displayDigits)

{

int digit, segment ;

int index, d, segVal ;

piHiPri (50) ;

for (;;)

{

for (digit = 0 ; digit < 6 ; ++digit)

{

for (segment = 0 ; segment < 7 ; ++segment)

{

d = toupper (display [digit]) ;

/\*\*/ if ((d >= '0') && (d <= '9')) // Digit

index = d - '0' ;

else if ((d >= 'A') && (d <= 'F')) // Hex

index = d - 'A' + 10 ;

else

index = 16 ; // Blank

segVal = segmentDigits [index \* 7 + segment] ;

digitalWrite (segments [segment], segVal) ;

}

digitalWrite (digits [digit], 1) ;

delay (20) ;

digitalWrite (digits [digit], 0) ;

}

}

}

/\*

\* setup:

\* Initialise the hardware and start the thread

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void setup (void)

{

int i, c ;

// 7 segments

for (i = 0 ; i < 7 ; ++i)

{ digitalWrite (segments [i], 0) ; pinMode (segments [i], OUTPUT) ; }

// 6 digits

for (i = 0 ; i < 6 ; ++i)

{ digitalWrite (digits [i], 0) ; pinMode (digits [i], OUTPUT) ; }

strcpy (display, " ") ;

piThreadCreate (displayDigits) ;

delay (10) ; // Just to make sure it's started

// Quick countdown LED test sort of thing

c = 999999 ;

for (i = 0 ; i < 10 ; ++i)

{

sprintf (display, "%06d", c) ;

delay (400) ;

c -= 111111 ;

}

strcpy (display, " ") ;

delay (400) ;

#ifdef PHOTO\_HACK

sprintf (display, "%s", "123456") ;

for (;;)

delay (1000) ;

#endif

}

/\*

\* funTime:

\* Jeremy Hillary Boob Ph.D. is a fictional character in the animated movie

\* Yellow Submarine.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void funTime (void)

{

char \*message = " feedbeef babe cafe b00b " ;

int i ;

for (i = 0 ; i < strlen (message) - 4 ; ++i)

{

strncpy (display, &message [i], 6) ;

delay (200) ;

}

delay (1000) ;

for (i = 0 ; i < 3 ; ++i)

{

strcpy (display, " ") ;

delay (150) ;

strcpy (display, " b00b ") ;

delay (250) ;

}

delay (1000) ;

strcpy (display, " ") ;

delay (1000) ;

}

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main:

\* Let the fun begin

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int main (void)

{

struct tm \*t ;

time\_t tim ;

wiringPiSetup () ;

mcp23s17Setup (200, 0, 0) ;

setup () ;

funTime () ;

tim = time (NULL) ;

for (;;)

{

while (time (NULL) == tim)

delay (5) ;

tim = time (NULL) ;

t = localtime (&tim) ;

sprintf (display, "%02d%02d%02d", t->tm\_hour, t->tm\_min, t->tm\_sec) ;

delay (500) ;

}

return 0 ;

}

# Interfacing Dot Matrix LED with Raspberry Pi

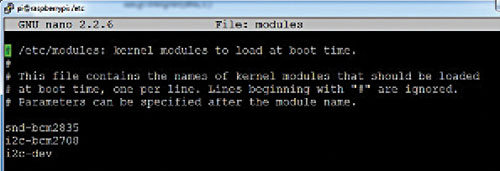
<https://electronicsforu.com/electronics-projects/hardware-diy/interfacing-dot-matrix-led-raspberry-pi>

You will find in this article projects for interfacing an 8×8 and a 5×7 dot-matrix display with Raspberry Pi -- Somnath Bera

Since Raspberry Pi does not have enough GPIOs, we have used its I2C bus to connect to MCP23017 IO expander to have 16 additional GPIOs for connecting to LED dot-matrix display such as 8×8 or 5×7. Pin 3 (SDA) and pin 5 (SCL) are I2C pins on Raspberry Pi board. Some libraries in the form of a GPL Python script, taken from Adfruit.com, are used in this project to manage the extended GPIOs.

First, you will see how a single 8×8 LED matrix is interfaced with GPIOs through MCP23017. Then you can expand the project to interface more dot-matrix displays.

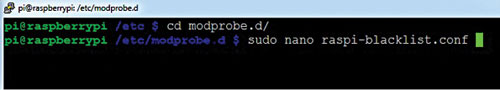
**System set up**  
To use I2C on Raspberry Pi, you need to enable a few things in Raspbian Wheezy operating system as these are not enabled by default. This is a fairly easy process. First you edit the modules file using the command:



 [stextbox id=”grey”]$ sudo nano /etc/modules[/stextbox]

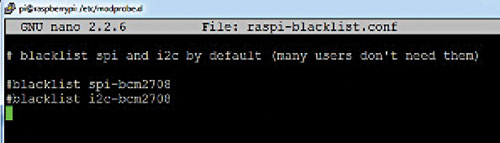
and add the following two lines (also refer Fig. 1):

 [stextbox id=”grey”]i2c-bcm2708  
i2c-dev[/stextbox]

Fig. 2: Editing the raspi-blacklist.conf file

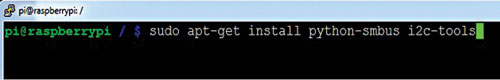
Save the file and exit nano. Next, open the raspi-blacklist.conf file by writing the command (also refer Fig. 2):

 [stextbox id=”grey”]$ sudo nano /etc/modprobe.d/  
raspi-blacklist.conf[/stextbox]

Fig. 3: Commenting the code

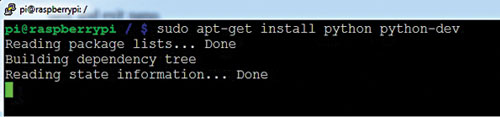
Comment out the code by putting # symbol at the beginning of the two lines as given below (also refer Fig. 3) and then save and exit.

 [stextbox id=”grey”]#blacklist spi-bcm2708  
#blacklist i2c-bcm2708[/stextbox]

Fig. 4: Install the Python-smbus and I2C tools

Now install the necessary software to manipulate the I2C bus using following commands (also refer Fig. 4):

 [stextbox id=”grey”]$ sudo apt-get update  
$ sudo apt-get install python-smbus  
i2c-tools[/stextbox]

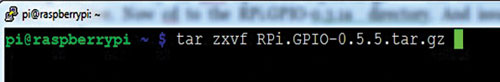
Fig. 5: Installing Python packages

Once the software is installed, reboot your Raspberry Pi so that these effects take place permanently. Also install Python and python-dev on Raspberry Pi using following commands (also refer Fig. 5):

 [stextbox id=”grey”]$ sudo apt-get install  
python python-dev[/stextbox]

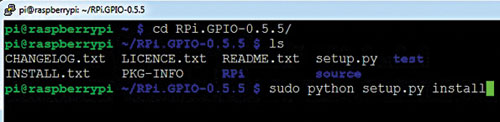
For interacting with the GPIO through Python, you have to install the GPIO Python libraries. The latest library available is RPi.GPIO-0.5.5.tar.gz. Download the same from the link below:

http://raspberry-gpio-python.googlecode.com/files/

Fig. 6: Extracting the tar file

Unzip or extract the tar file by using command (also refer Fig. 6):

 [stextbox id=”grey”]tar zxvf RPi.GPIO-  
0.5.5.tar.gz[/stextbox]

Fig. 7: Installing the setup.py

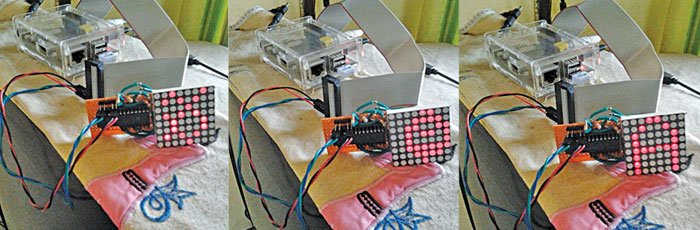
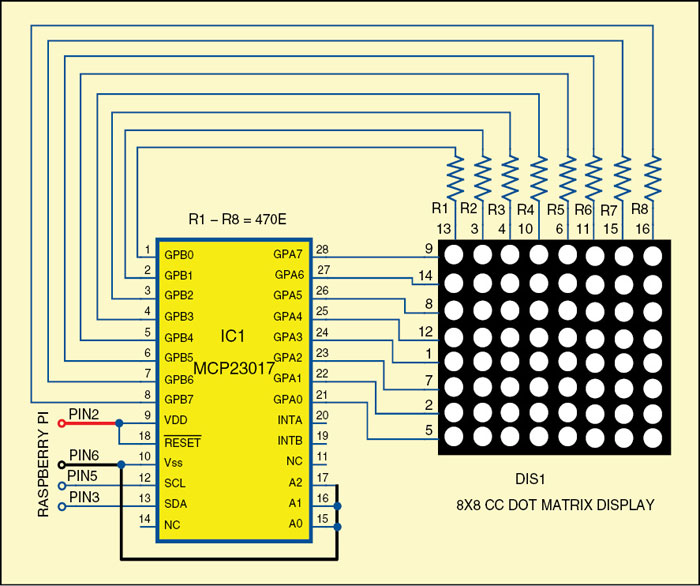
Now change directory to the RPi.GPIO-0.5.5 and write the following command (also refer Fig. 7):

 [stextbox id=”grey”]$ sudo python setup.py  
install[/stextbox]

The command will install Raspberry Pi GPIO library into Python. Now reboot the Raspberry Pi using command:

 [stextbox id=”grey”]$ sudo reboot[/stextbox]

**Project 1: Multiplexing of 8×8 dot-matrix display**  
In this project, you will use two separate files—one to create a template of figures on the 8×8 dot- matrix (using ‘1’s) and the other to control the GPIOs for drawing figures on the LED matrix. So whatever letter or character you want to display, make the changes in letter.py code and the same will be drawn on the LED matrix.

Fig. 8: Multiplexing with GPIOs using 8×8 dot-matrix display Fig. 9: Circuit of 8×8 dot-matrix display

Make the connections as per the circuit shown in Fig. 9. MCP23017 (28-pin DIP) is used here as it has 16 GPIOs—8 for rows and 8 for columns of 8×8 LED matrix. Remember to use current-limiting resistors, whose values can be between 470 ohms and 1-kilo-ohm, in series with the LEDs.

Four wires are connected to the Raspberry Pi—two for I2C and two for 5V supply and ground connections.

Before testing, make sure that 5V supply is going to the correct pin in Raspberry Pi board, else it will heat up the microprocessor and damage it. Also ensure the reset pin 18 of MCP23017 is connected with +5V rail.

**Circuit and working**

To test the I2C connection on Raspberry Pi, open the terminal and issue the following command:

 [stextbox id=”grey”]$ sudo i2cdetect -y 0  
or  
$ sudo i2cdetect -y 1[/stextbox]

If your Raspberry Pi is version 1 then use 0, else 1, and watch the screen shown in Fig. 10 appear at the terminal prompt.

Fig. 10: Testing I2C connection

Congratulations! Your Raspberry Pi has successfully detected the hardware at 0x20 address. The address is 0×20 because all the three address pins (A0, A1, A2) of MCP23017 are set low. If you had set only A0 high (connected to +5V) the address would have become 0×21.

This signifies one fundamental issue with I2C communications: You can connect any number of devices on the I2C bus (pin 3 and pin 5) but all devices should have unique addresses. So, setting A0, A1 and A2 pins will define different addresses for different devices. You can also control the device by issuing appropriate command. You may try this as explained below to gain more confidence.

There is another way of testing the I2C connection on Raspberry Pi. Connect an LED on A0 pin and issue the following commands:

 [stextbox id=”grey”]$ sudo i2cset 0x20 0x00 0x00 # this  
will switch on the LED on A0  
$ sudo i2cset 0x20 0x00 0x01 # this  
will switch off the LED on A0[/stextbox]