Bien que la gestion d'afficheurs 7 segments s'apparente à du pilotage banal de LED, il s'en différentie par deux facettes propres à ce type d'applications. D'une part il faut disposer d'une table de transcodage entre les caractères à symboliser et les diodes électroluminescentes à illuminer sur les afficheurs. D'autres part le nombre important de LED à piloter dépassant celui des sorties disponibles sur le microcontrôleur et il faut avoir recours à du multiplexage. L'afficheur utilisé dans cette application est un SH5461AS facilement disponible dans le commerce actuellement. Tout modèle à cathodes communes conviendra. La Fig.2 précise la désignation standard des divers éléments d'un afficheur 7 segments banal. Le brochage du SH5461AS est présenté sur la Fig.3 sur laquelle **K1** à **K4** désigne les cathodes communes des divers chiffres pris dans l'ordre de la gauche vers la droite. Les douze broches sont toutes utilisées, c'est le nombre d'E/S qui seront mobilisées sur Arduino pour piloter entièrement cet afficheur. Sur la Fig.3 le composant est représenté vu de dessus comme pour un circuit intégré de type DIL. Électroniquement deux approches sont possibles pour interfacer le SH5461AS. La première consiste, comme montré sur la Fig.4 à porter au potentiel de la masse les cathodes communes avec les sorties de pilotage d'Arduino, et à alimenter en +5v les segments à travers une résistance de limitation d'1kΩ par exemple. Cette approche correspond à celle d'une application réelle. Elle présente l'avantage d'avoir une bonne luminosité de l'afficheur, et surtout chaque segment présente une clarté égale et constante quel que soit le symbole représenté. Si les huit éléments sont allumés, l'intensité que doit fournir la sortie d'Arduino qui pilote **Ki** atteind 40 mA. Ce courant correspond au maximum toléré par une sortie binaire. Cette solution reste donc compatible avec les limitations de l'ATmega328 et n'impose pas de transistor de commutation par exemple. Par contre, dans le cas d'une simple évaluation, il est tout à fait envisageable de simplifier le montage expérimental. La Fig.5 décrit le schéma optimisé dans lequel on n'utilise que quatre résistances au lieu de huit. Chaque résistance est intercalée dans le pilotage des cathodes communes. De ce fait le courant maximal pour l'ensemble du "DIGIT" ne dépassera pas 5mA. Ce courant va se répartir dans les divers segments allumés. De ce fait, la luminosité de l'afficheur va varier de façon sensible en fonction du nombre de LED actives. Quand le chiffre huit est représenté et que le point décimal est allumé, nous obtenons la luminosité minimale. Elle est maximale quand un seul élément s'éclaire. Pour ne pas surcharger ce segment ou le Dp, on limite le courant total aux 5mA annoncés. Compte tenu du très bon rendement des LED de l'afficheur, cette solution simplifiée se justifie pleinement et s'avère très largement suffisante pour développer le programme. Que l'on adopte la solution de la Fig.4 ou celle de la Fig.5 dans les deux cas le programme reste strictement le même. Seuls les courants mis en jeu seront différents en fonction du choix effectué. Le plus économe est naturellement le schéma de la Fig.5 qui en moyenne consomme un courant quatre fois plus faible. Par contre, dans une application pérenne il semble évident que le montage de la Fig.4 s'imposerait.

Fig.1

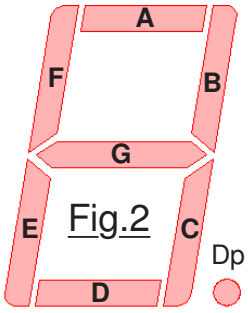


Fig.2

Brochage du SH5461AS vu de dessus.

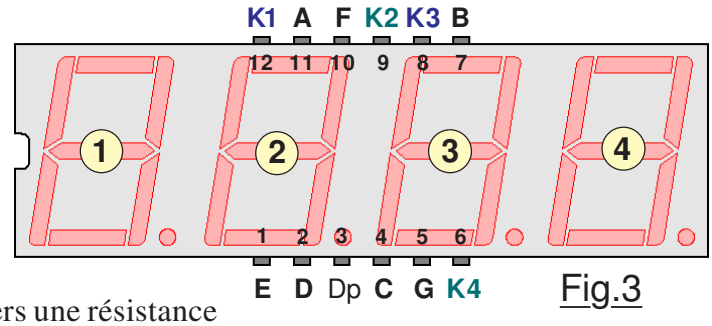


Fig.3

Fig.4

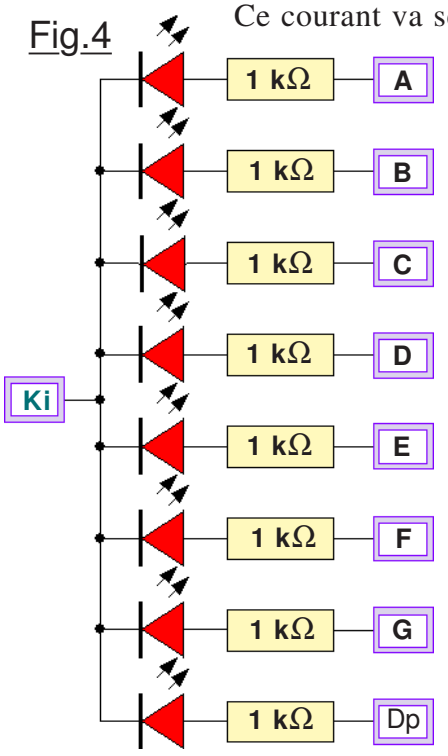
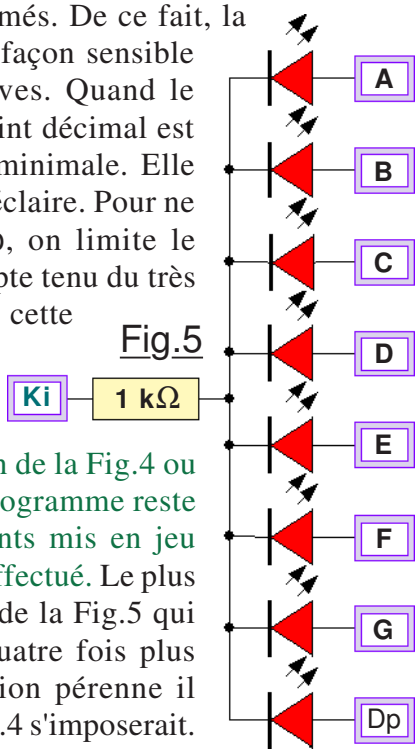


Fig.5



Analysons les détails d'agencement du programme Journal_defilant_7_segments.ino qui émule une petite application "ludique". Il s'agit d'un journal défilant. Le texte proposé est frappé dans la fenêtre du terminal série virtuel d'Arduino. Ce texte peut faire jusqu'à 63 caractères ce qui correspond au maximum possible dans la "fenêtre de saisie de la ligne USB". Lorsque l'on valide la ligne de texte saisie sur le terminal virtuel, celle-ci s'affiche par décalage de droite à gauche sur le SH5461AS. La vitesse de décalage est gérée par le programme. Le défilement latéral s'arrête sur les quatre derniers caractères des divers textes proposés. La nouvelle phrase "pousse la dernière". Compte tenu de la faible définition d'un afficheur 7 segments pour symboliser les caractères, il faut avoir recours à un mixage de lettres majuscules et de lettres minuscules pour pouvoir représenter tout l'alphabet. Le "W" , le "X" et "Z" ne sont pas très beaux, mais avec un peu d'habitude et dans le contexte ils se lisent facilement.

Le principe du multiplexage : La technique consiste à placer sur les 8 éléments *(Segments et Dp)* le niveau électrique haut d'environ +5v. Puis on allume l'afficheur concerné en portant sa cathode **Ki** à la masse. On laisse l'afficheur allumé durant 5mS puis on ramène sa cathode **Ki** au +5v pour l'éteindre. On passe immédiatement à l'afficheur voisin et on recommence. La rapidité de rafraichissement et l'inertie visuelle donnent l'illusion d'un éclairement permanent. Les mesures au périodemètre numérique donnent les valeurs suivantes : Allumage durant 5 mS, extinction pendant 15,2mS *(Balayage des trois autres "digits")* avec une période de 20,2mS qui se traduit par une fréquence de multiplexage légèrement inférieure à 50Hz. Concrètement c'est la fréquence de balayage de la boucle principale de base void **loop**. Cette boucle infinie représentée sur la Fig.6 passe la majorité du temps dans le multiplexage. Le traitement colorié par coloriage en rose sur l'organigramme n'exige que quelques milli secondes et passe totalement inaperçu sur le visuel. Le TAMPON comporte huit emplacements comme représenté en jaune sur la Fig.7 avec le FIFO colorié en bleu pastel. Le FIFO dispose de 63 octets pour pouvoir loger la ligne saisie sur le terminal série dont la mémoire de stockage correspond à cette taille. Tout caractère suivi d'un point décimal sera considéré comme un seul "DIGIT" puisqu'ils sont associés sur le SH5461AS. C'est la raison pour laquelle sur la Fig.7 les deux octets associés au même "DIGIT" sont mis en évidence par couleur orange. Comme rien n'interdit de frapper des caractères tous séparés par des points décimaux, il faut donc prévoir huit octets dans le TAMPON. Le balayage de multiplexage prend en compte tous les octets non nuls *($00 : Sentinelle)* dans le TAMPON, sachant qu'au maximum il ne peut y contenir que quatre caractères avec éventuellement un point décimal associé. Le transfert du FIFO vers le TAMPON en (2) est représenté sur la Fig.7 avec des marqueurs d'emplacements vides constitués d'octets nuls. Tous les caractères ne sont pas visualisables sur un afficheur 7 segments. Par exemple des caractères comme "#" et "%" seront ignorés. Lors du filtrage en (1) ces caractères ignorés sont remplacés par des "*" qui sont également non affichables. Ces caractères seront alors représentés en allumant uniquement le segment **D** pour les distinguer de l'espace qui lui éteint entièrement le "DIGIT" correspondant. Le transcodage caractère vers "DIGIT" utilise un octet avec les segments affectés comme montré ci-contre, les éléments étant ordonnés de la gauche vers la droite.

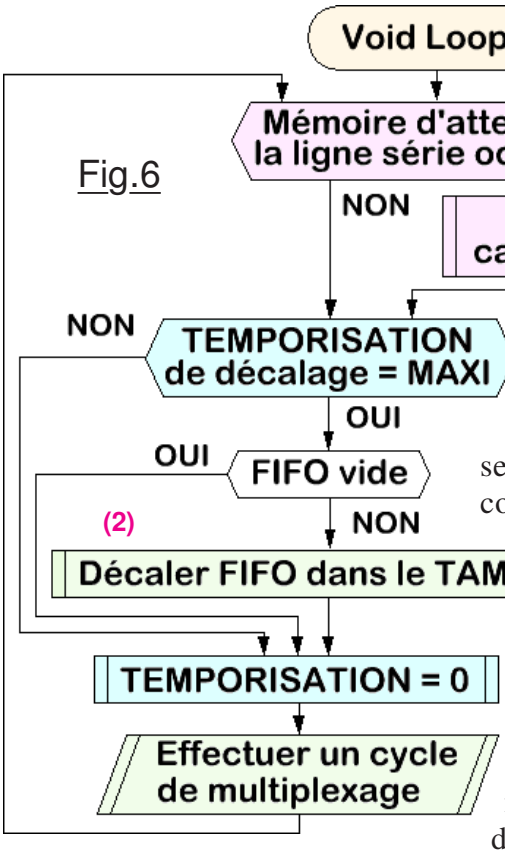


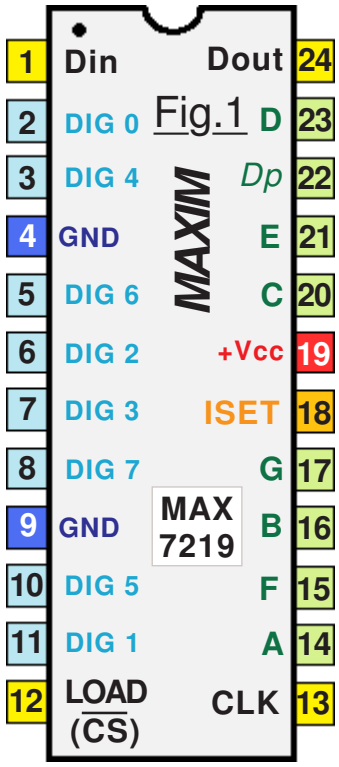
Fig.6

Fig.7 (2)

A	B	C	D	E	F	G	Dp

T	A	M	P	.	00	00	00	O	N	F	I	F	O	00	00	00

Toute application électronique exigeant un grand nombre de composants discrets pour sa mise en œuvre finit par imposer un ou des circuits intégrés spécialisés. Le multiplexage d'afficheurs 7 segments ou 16 segments n'échappe pas à cette règle imposée par les exigences commerciales. Le circuit MAX 7219 dont le brochage est donné en Fig.1 permet de multiplexer 8 afficheurs de type 7 segments à cathodes communes. Il peut également servir pour le multiplexage de matrices de LED très courantes sur le marché. Ce circuit intégré complexe se pilote par une ligne de type SPI, plusieurs de ces circuits pouvant être mis en "cascade" pour augmenter le nombre de "DIGIT" alignés sur le dispositif d'affichage. Une seule résistance **R** de 47 kΩ pour limiter le courant est suffisante. *(On peut descendre à 10kΩ mais ce n'est pas recommandé)* Montré sur la Fig.2 elle se place entre le **+Vcc** et **ISET**.



Caractéristiques techniques du circuit MAX7219 :

- Alimentation : 4,0 à 5,5 Vcc.
- Consommation en veille : 150 µA.
- Consommation tout éteint : 8 mA.
- Consommation tout allumé : 330 mA.
- Fréquence de balayage : 700 Hz à 1300 Hz. (1)

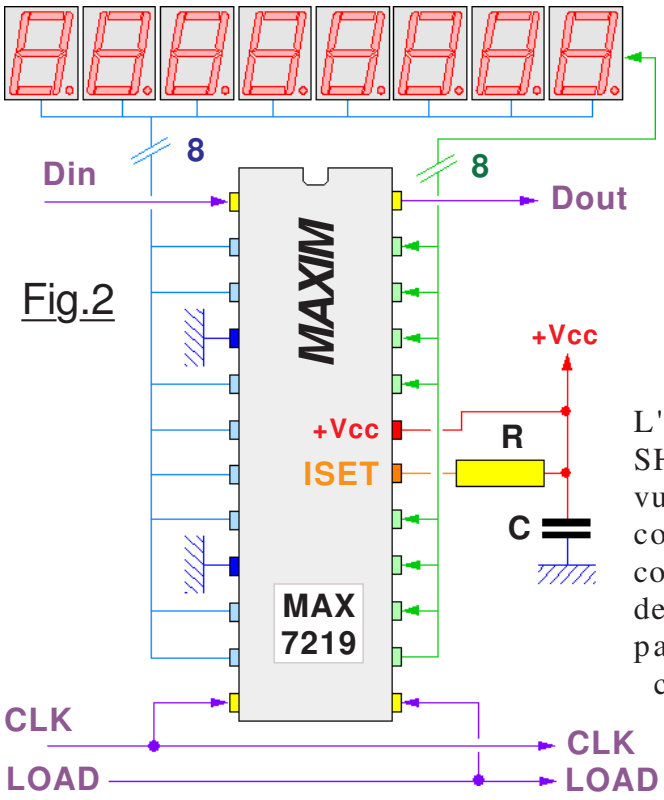


Fig.2

Brochage du SH5461AS vu de dessus.

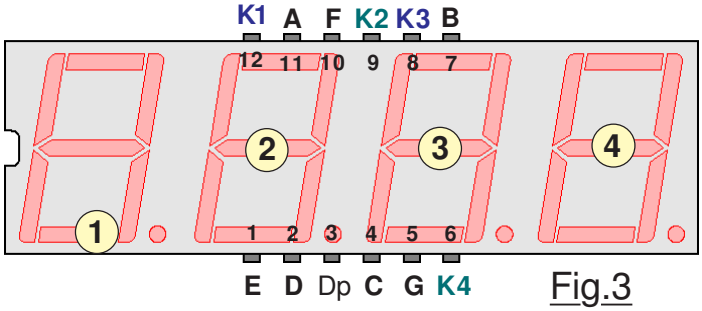


Fig.3

L'application présentée ici utilise un afficheur SH5461AS dont le brochage est donné sur la Fig.3 en vue de dessus. Il regroupe quatre "DIGIT" à cathodes communes. Il est impératif de mettre en place le condensateur **C** d'environ 0.01µF ou le circuit présente des aléas de fonctionnement. Dès que le courant exigé par les afficheurs dépasse les possibilités du composant, il ne fonctionne pas correctement et les "DIGIT" restent ± noirs. Il est préférable d'insérer une résistance de 47 kΩ quitte à limiter le balayage à quatre "DIGIT" pour rétablir une forte luminosité. Le programme Essai_AFF_7_SEGmt.ino donne un exemple très simple de mise en œuvre d'une petite application d'affichage sur 7 segments par utilisation de la bibliothèque **LedControl.h** qui fournit les fonctions de base pour gérer la ligne de commande SPI. *(Et le codage Binaire vers 7 SEGMENTS pour symboliser facilement les chiffres)* Dans cet exemple le balayage est limité aux quatre "DIGIT" et l'on se trouve aux limites de courant acceptable par le circuit intégré. La fréquence de balayage mesurée se situe aux environs de 1300 Hz. Si on valide le balayage sur tous les afficheurs elle chute à 700 Hz. Le courant moyen mesuré est de 33 mA. Le programme permet, au moyen d'un potentiomètre dont la tension variable entre 0 et +5Vcc branchée sur l'entrée analogique A0, de faire varier progressivement la luminosité de l'affichage qui sur RESET est minimal. On constate que lorsque la lumière est maximale, de petits et courts aléas apparaissent furtivement. *(Courant limite de courts instants)*

(1) : La fréquence de balayage est directement fonction du nombre de DIGIT balayés.

Gérer un grand nombre d'afficheurs 7 segments ou un nombre important de LED ne se conçoit plus sans le secours d'un circuit intégré de multiplexage tel que le MAX 7219 déjà mis en œuvre dans l'application précédente. Des matrices de LED existent à profusion dans le commerce, dont les dimensions sont variables. Parmi les plus courantes actuellement on trouve le module 1088AS par exemple dont la popularité à généré des kits "clef en main" tel que celui de la Fig.1 facilitant grandement la concrétisation de petites applications. Le 1088AS comporte 64 LED arrangée en huit lignes et huit colonnes. La Fig.2 en présente le brochage lorsque ce composant est vu par le dessus. Les broches sont numérotées comme celles d'un circuit intégré de type DIL, toujours observé par dessus. Le petit montage électronique de la Fig.1 tient compte des caractéristiques de cet afficheur matriciel. Comme montré sur la Fig.3 les connecteurs sont conformes à son brochage interne. Notons au passage que si l'espacement entre deux broches sur le 1088AS fait bien les 2,54mm standard, la distance entre les deux lignes de broches n'est pas normalisée puisqu'elle fait 24,5mm ce qui ne correspond pas à exactement un pouce.

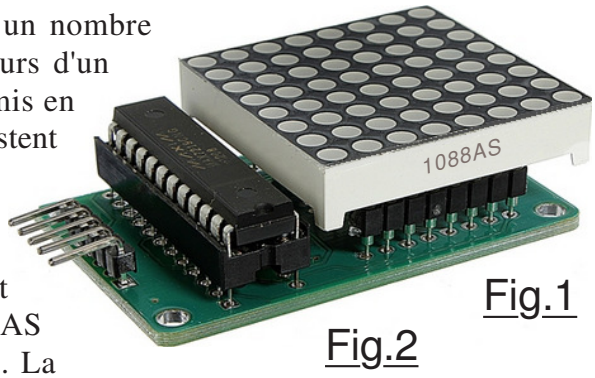
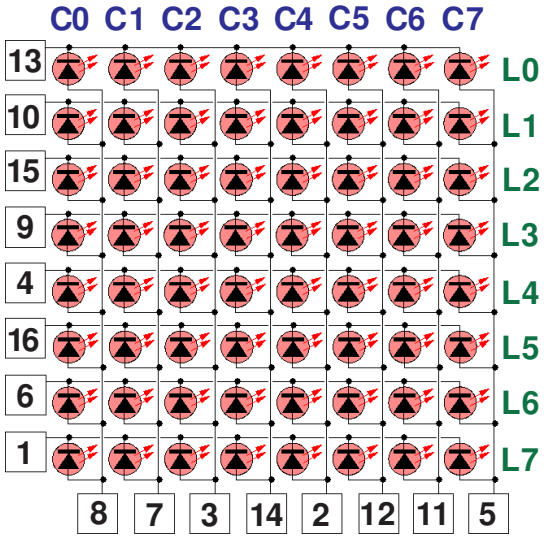


Fig.1

Fig.2



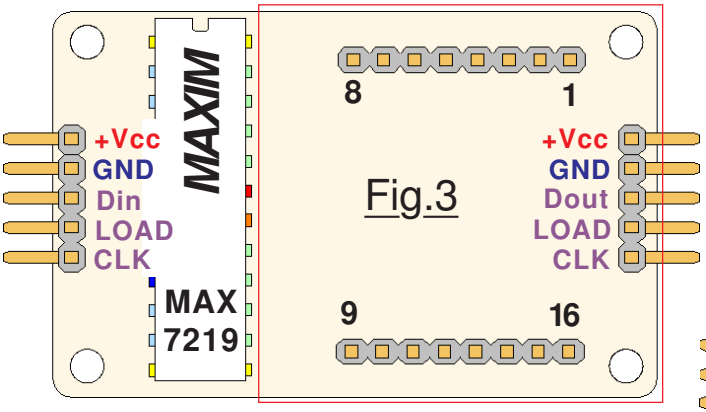
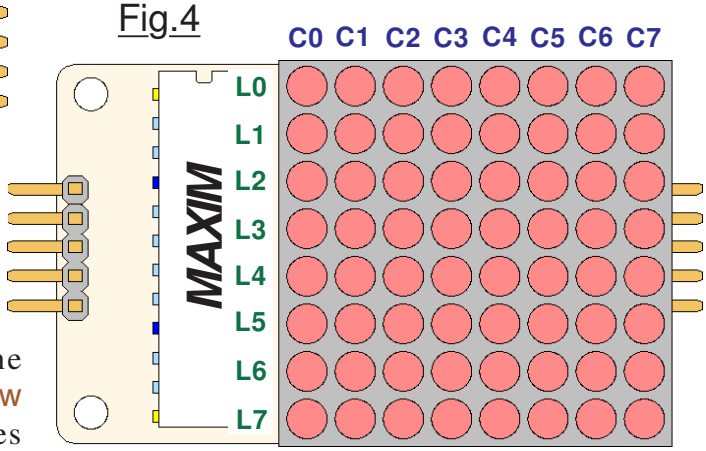


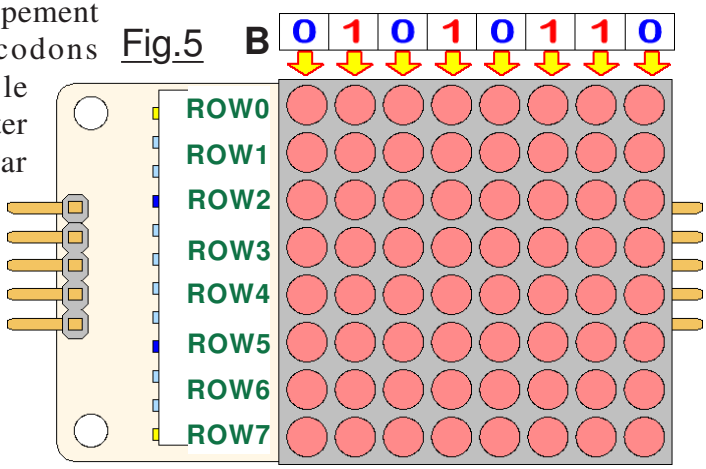
Fig.3

Fig.4



La bibliothèque **LedControl.h** permet de gérer l'allumage des LED soit individuellement soit par "Rangées". Pour programmer facilement une application en usant des procédures setLed et setRow il importe de faire le lien entre les coordonnées indiquées et la position de l'afficheur sur le module électronique. La Fig.4 précise l'ordre des lignes et des colonnes quand on programme LED par LED avec setLed en précisant leurs coordonnées. La Fig.5 pour son compte précise l'ordre des "rangées" pour setRow avec le symbole de l'OCTET dans lequel on précise par des "1" et des "0" l'état des LED dans le groupement traité. Mais comme nous codons librement les coordonnées ou le contenu des octets, on peut orienter l'afficheur comme on le désire. Par exemple, pour le petit programme l'application **Essai_Matrice_LED_8x8.ino** le circuit imprimé est orienté comme montré sur la Fig.6 de façon à ne pas être masqué par les fils de liaison pour l'ISP.

Fig.5



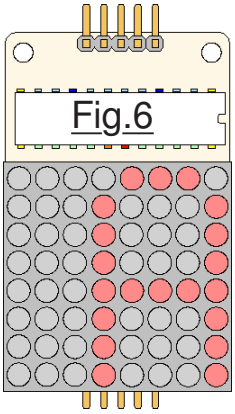


Fig.6

Piloter un moteur pas à pas relève d'une logique élémentaire et ne présente pas de difficulté particulière. Il importe toutefois d'interfacer le microcontrôleur avec une électronique de commutation apte à gérer la tension d'alimentation des bobines, le courant consommé et les surtensions de commutation. Compte tenu de l'importance de ce type de moteurs dans les automatismes, les produits commerciaux couvrent tous les besoins, que ce soit avec des moteurs de toutes puissances, ou avec des circuits intégrés spécialisés pour les piloter. L'un des plus populaires étant l'ULN2003AN qui simplifie considérablement l'interface de commutation de puissance. Le petit moteur utilisé dans cette application montré sur la Fig.1 est un 28BYJ-48 avec en Fig.2 le brochage de son connecteur.

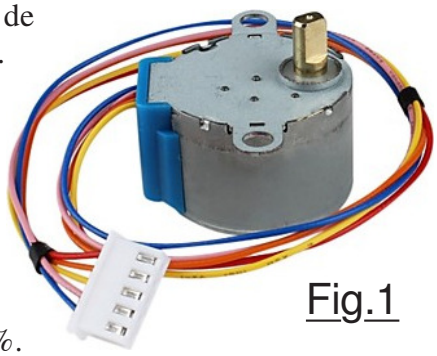
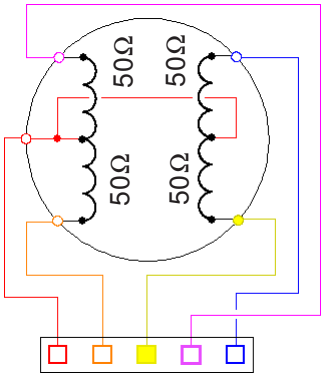


Fig.1

Caractéristiques techniques du moteur 28BYJ-48 :

Fig.2



- Tension d'alimentation : 5Vcc.
- Résistance en courant continu : 50Ω ± 7%.
- Résistance d'isolement >10MΩ. *(500V)*
- Rapport de réduction: 1/64.
- Angle de rotation par pas: 5,625° / 64. *(Soit 0,08789°/pas)*
- Fréquence d'utilisation : 100Hz.
- Diamètre 28 mm, épaisseur 20mm et implantation 35 mm.

Les moteurs pas à pas bipolaires comportent généralement quatre fils sortant de ces derniers plus éventuellement un commun "central", contrairement aux moteurs unipolaires qui n'ont aucun lien de centre commun. Le modèle utilisé 28BYJ-48 est bipolaire avec fonctionnement à huit phases.

Le pilotage d'un moteur pas à pas :

Le schéma fondamental d'une électronique de pilotage est développé en Fig3 sur lequel on retrouve un transistor par bobine à commuter. Pour gérer un moteur bipolaire avec commun central il faut donc deux sections analogues à celle proposée ci-contre. On retrouve en **Rb** la résistance de limitation du courant de base, et surtout les diodes **D** qui éliminent les surtensions inductives. On peut éventuellement augmenter le gain en courant en complétant **T1** *(Et T2)* par un deuxième transistor agencé en Darlington. La chute de tension à l'état saturé avoisine 1Vcc.

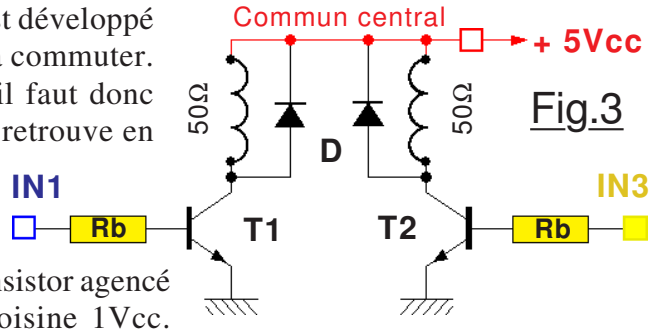


Fig.3

Alimenté sous 5V, chaque enroulement de 50Ω va consommer environ 100mA. Dans le montage d'expérimentation les transistors sont des composants ordinaires pour faible puissance. Leur gain en courant étant d'environ 100, pas besoin de les assembler en Darlington. Avec une résistance de base d'1kΩ ils sont en saturation. Ils ne chauffent absolument pas, donc inutile de les placer sur radiateurs. Les diodes **D** sont des composants pour petits signaux banales. Chaque collecteur est relié à une LED qui va au +5Vcc avec une résistance de limitation de courant également d'1kΩ pour visualiser l'état passant des transistors sur chaque bobine et également pouvoir contrôler le mode veille. Quand le moteur est en rotation et les quatre LED allumées, le courant total fait environ 120mA ce qui est largement supportable par une alimentation USB de la carte Arduino. Le petit programme Moteur_PAS_A_PAS.ino permet de tester l'interface de composants discrets représentée en Fig.3 qui utilise deux fois ce petit schéma.

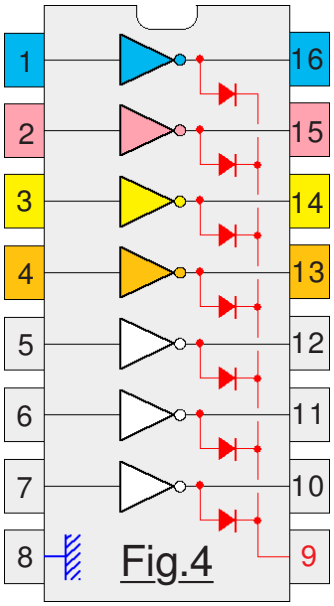


Fig.4

Le circuit intégré de pilotage ULN2003 :

Bien que le nombre de composants pour gérer la commutation du moteur pas à pas bipolaire ne soit pas très élevé, il est bien plus avantageux d'utiliser un circuit spécialisé dont le coût n'est pas plus élevé que celui de la solution intégrant des composants discrets et qui tient bien moins de place sur

le circuit imprimé. Un ULN2003 dont le brochage est dévoilé sur la Fig.4 intègre sept circuits de commutation indépendants schématisés sur la Fig.5 et pourvus des diodes de protection sur les entrées, ainsi que les diodes **D** toutes réunies à un Commun. Toujours avec le programme Moteur_PAS_A_PAS.ino il suffit de remplacer les composants discrets par un tel circuit intégré pour obtenir immédiatement le même comportement du moteur. Mis à part éventuellement les DEL et leurs résistances de limitation de courant, il n'y a rien à ajouter au circuit intégré pour piloter le 28BYJ-48. On remarque sur la Fig.5 que la résistance de base **Rb** est intégrée, ainsi que ***D1*** pour protéger l'entrée d'une tension négative et ***D2*** pour éviter un courant inverse sur la sortie. Le montage adopté est celui d'un amplificateur de courant en montage Darlington. La tension résiduelle en état de saturation est inférieure à 1Vcc, du coup le courant total passe à 170mA. De plus le moteur est plus "nerveux" et il devient possible de passer delay(2) à delay(1) pour obtenir une rotation plus rapide sans pour autant perdre des pas. Sachant que le collecteur peut commuter 500mA et que le courant de base peut atteindre 25mA en permanence, dans notre application l'ULN2003 n'est vraiment pas très sollicité.

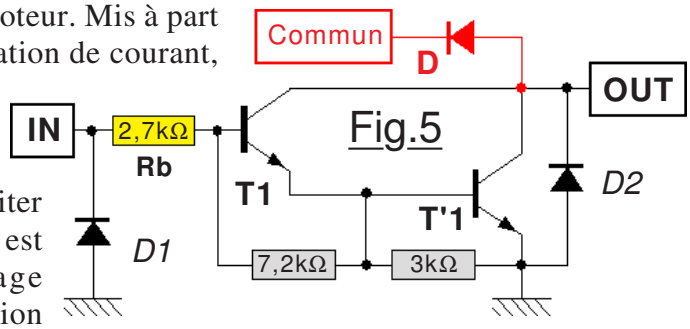


Fig.5

Carte électronique du commerce :

Compte tenu de la forte demande de tel circuits pour des petites applications de robotique, on se doute qu'il existe dans le commerce une foule de cartes toutes câblées nous épargnant l'étude et la réalisation d'un circuit imprimé. Leur coût restant très modéré, les utiliser relève pratiquement de l'évidence. La Fig.6 montre l'un de ces produits dont le connecteur est directement compatible avec le branchement de la fiche d'un 28BYJ-48. De plus, ce tout petit circuit intègre quatre LED de visualisation avec leurs résistances de limitation en courant. Le "strap" visible sur la Fig.6 permet d'isoler le moteur du Commun

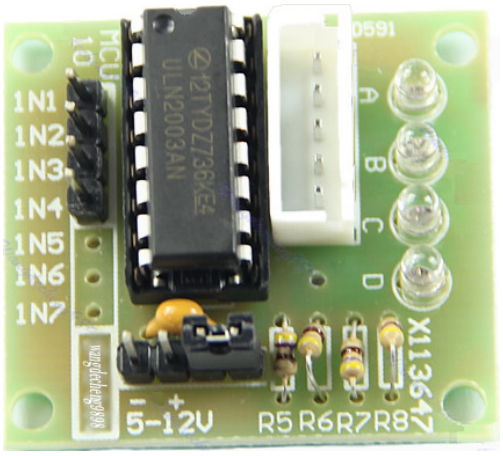


Fig.6

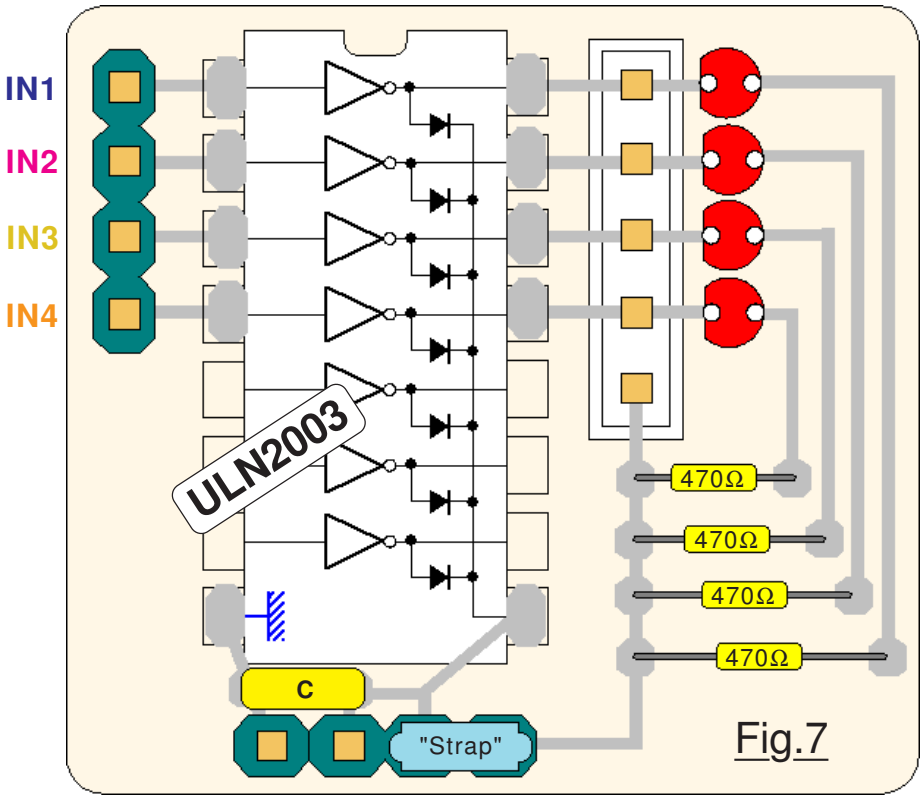


Fig.7

ce qui permet à la demande de commuter aussi des moteurs unipolaires. Le dessin de la Fig.7 ci-contre présente le câblage de la petite carte électronique du commerce. On notera la présence d'un condensateur de découplage **C** entre le +Vcc et la masse. Les trois opérateurs non utilisés ne sont pas reliés aux connecteurs et de ce fait sont indisponibles. Le circuit intégré est placé sur un support pour en faciliter l'éventuel changement bien que ce composant soit très fiable. La tension d'alimentation préconisée par la sérigraphie précise que l'on peur aller jusqu'à +12Vcc si le moteur utilisé est conçu pour cette tension.

L'utilisation des moteurs pas à pas n'est pas très compliquée. On peut définir ses propres procédures comme dans le programme Moteur_PAS_A_PAS.ino cité en exemple en page 60. Il est naturellement possible d'utiliser la bibliothèque Stepper.h décrite en page 7 du livret consacré aux librairies les plus courantes. Le programme est bien plus simple à écrire, et plus facile à lire. Mais c'est au détriment de la taille occupée en mémoire. L'utilisation de cette bibliothèque consomme environ 570 octets de plus que le même programme Sans_Librairie.ino écrit intégralement avec des procédures personnelles. Pour montrer la facilité d'utilisation de la librairie spécialisée, le programme Librairie_PAS_A_PAS.ino listé ci-dessous permet de piloter un 28BYJ-48 commandé par un potentiomètre dont la sortie va sur l'entrée analogique A0. Noter qu'il n'est pas nécessaire d'aller télécharger Stepper.h sur Internet car elle fait partie des librairies déjà disponibles par défaut à l'installation du compilateur.

```
/* Test des moteurs pas à pas en utilisant la bibliothèque Stepper.h décrite sur :
   http://www.arduino.cc/en/Reference/Stepper
   Un potentiomètre est branché sur l'entrée analogique A0. Le moteur pas à pas
   recopie linéairement les variations de tension présentes sur A0. */

#include <Stepper.h>

#define Nb_tours_par_minute 240 // Vitesse de rotation imposée au moteur.
#define Nb_pas_par_tour 64 // Définition du moteur réducteur non pris en compte.

// Déclaration des brachements.
#define Broche1 9  // Bleu   (broche 1)
#define Broche2 11  // Rose   (broche 2)
#define Broche3 8  // Jaune  (broche 3)
#define Broche4 10  // Orange (broche 4)

Stepper MOTEUR(Nb_pas_par_tour, Broche3, Broche1, Broche4, Broche2);

int Mesure_sur_A0, A0_Precedente = 0;

void setup() {
  MOTEUR.setSpeed(Nb_tours_par_minute); // Définir la vitesse de rotation.
  Mesurer(); A0_Precedente = Mesure_sur_A0; } // Immobile au RESET.

void loop() {
  Mesurer();
  MOTEUR.step((Mesure_sur_A0 - A0_Precedente) * 2.05); @
  // Valeur multipliées par 2.05 pour que la portée du potentiomètre corresponde à
  // un tour effectué en sortie de réducteur sur le moteur 28BYJ-48.
  A0_Precedente = Mesure_sur_A0; }

void Mesurer() {
  Mesure_sur_A0 = analogRead(0); // 50 mesures pour stabiliser le résultat.
  for (byte I = 0; I < 50; I++) {Mesure_sur_A0 = (Mesure_sur_A0 + analogRead(0)) / 2; } }
```

Ce programme "recopie" la position de l'axe du potentiomètre. Toute modification de son ajustement se traduit par une rotation du moteur dans le sens direct ou dans le sens rétrograde pour respecter la consigne. L'instruction @ ajuste le nombre de pas de telle façon que la portée du potentiomètre *(De 0 à +5Vcc sur l'entrée A0)* corresponde à pratiquement un tour d'arbre en sortie du réducteur.

ATTENTION : Le moteur ne fonctionnera correctement que si l'on respecte les branchements électriques définis sur la Fig.1 ci-contre.

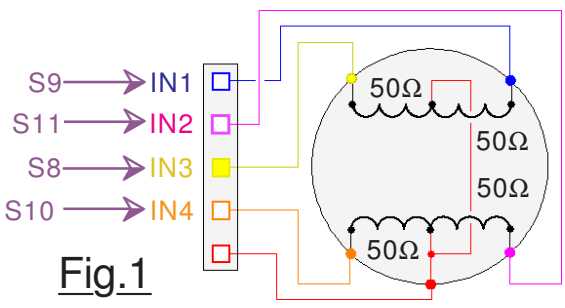
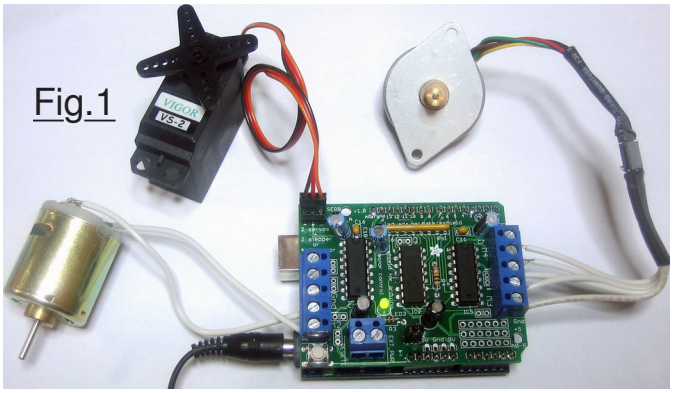


Fig.1

Shield ADAFRUIT pour piloter des moteurs à courant continu :

L'entreprise ADAFRUIT implantée à New York s'est spécialisée notamment dans la conception et la fourniture de petits modules pour les cartes Arduino et propose divers produits **dont un shield spécifique pour piloter des petits moteurs**. Comme montré sur la Fig.1, cette carte est très polyvalente, et l'on peut aussi-bien y brancher des moteurs à courant continu, des moteurs pas à pas ou des servomoteurs. Le nombre de moteurs pilotables simultanément dépend des types de motorisations utilisées.



Caractéristiques de la carte d'interface :

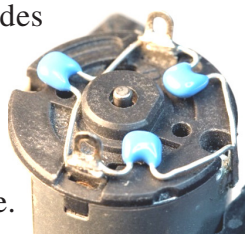
- Deux connexions pour des servomoteurs 5V. *(Liaisons directes sur les E/S d'Arduino.)*
- Quatre ponts en H gérés par des circuits intégrés L293D capables de fournir 0,6 A par pont *(1.2A crête)* pourvus d'une protection d'arrêt thermique et de diodes internes "de roue libre". Ces ponts peuvent fonctionner avec des tensions allant de 4.5Vcc à 25Vcc.
- Gère dans les deux sens de rotation possibles jusqu'à quatre moteurs à courant continu avec sélection individuelle de la vitesse sur 8 bits.
- Pilote jusqu'à deux moteurs pas à pas *(Unipolaires ou bipolaires)* avec une seule bobine, double bobinage ou enroulements entrelacés.
- Les résistances de charge des moteurs sont désactivées pendant la mise sous tension.
- Un bornier sérieux permet d'établir aisément les liaisons électriques avec les moteurs. *(18-26AWG)*
- Le bouton de réinitialisation d'Arduino déporté sur le circuit imprimé.
- Deux broches et un cavalier permettent d'isoler l'alimentation externe des moteurs de la logique.
- Possibilité de brancher simultanément deux servomoteurs sous 5Vcc et :
  4 moteurs à courant continu ou,
  2 moteurs pas à pas ou,
  2 moteurs à courant continu et un moteur pas à pas.

Entrées/Sorties d'Arduino utilisées :

- Les six broches des entrées analogiques sont totalement disponibles.
- Les deux broches des E/S numériques 2 et 13 sont totalement disponibles.

Les broches suivantes ne sont utilisées que si un moteur s'y trouve en cours d'utilisation :
Broche numérique 11:Moteur DC n ° 1 / Pas à pas n ° 1*(Activation / contrôle de vitesse)*
Broche numérique 3: Moteur DC n ° 2 / Pas à pas n ° 1*(Activation / contrôle de vitesse)*
Broche numérique 5: Moteur DC n ° 3 / Pas à pas n ° 2*(Activation / contrôle de vitesse)*
Broche numérique 6: Moteur DC n ° 4 / Pas à pas n ° 2*(Activation / contrôle de vitesse)*
Les broche numérique 4, 7, 8 et 12 sont utilisés pour commander les moteurs DC ou pas à pas via la logique de pilotage du circuit intégré 74HC595.
Les broches suivantes ne sont utilisées que si un servomoteur y est en cours d'utilisation :
Broche PWM numérique 9 : Commande du connecteur **SERVO_2**.
Broche PWM numérique 10 : Commande du connecteur **SER1**.

NOTE : Une LED s'allume *(Voir Fig.2)* si l'alimentation de puissance est effective. Si elle n'est pas allumée, les moteurs DC ou les moteurs pas à pas ne fonctionneront pas. Les ports des servomoteurs sont en 5Vcc et n'utilisent pas l'alimentation extérieure. Ce module ne peut pas piloter des moteurs à faible tension prévus pour 3Vcc. Il est prévu pour une tension de 6Vcc ou plus. Les moteurs DC génèrent beaucoup de parasites tant magnétiques qu'électriques pouvant perturber Arduino. Il est donc fortement conseillé de les alimenter par la ligne extérieure et de munir chaque moteur de trois condensateurs d'antiparasitage : Un condensateur en parallèle sur le moteur, deux entre les broches d'alimentation et la masse.



Alimentation des moteurs :

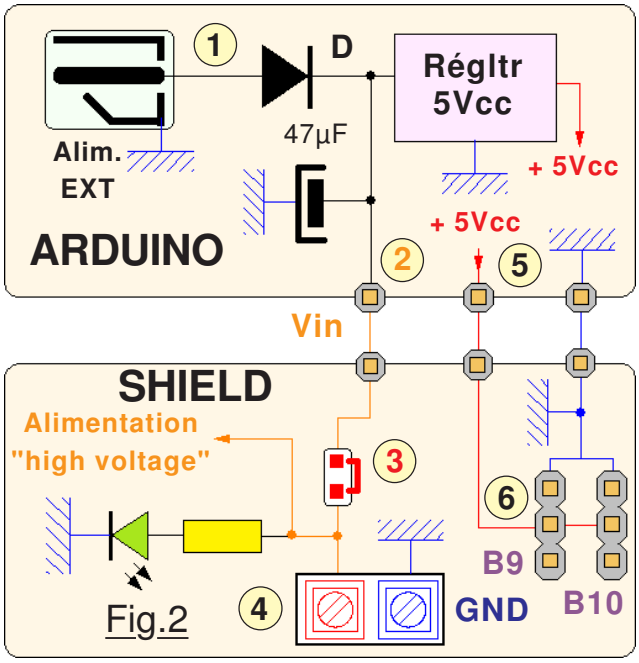


Fig.2

Si on désire n'utiliser qu'une seule source pour fournir l'énergie électrique à Arduino et au module de gestion des moteurs, il suffit de brancher une alimentation externe sur la prise prévue à cet effet **1**. Dans ce cas on n'utilise pas le connecteur **4** et le cavalier **3** doit être en place sur son connecteur. Cette méthode présente toutefois un certain nombre d'inconvénients. Le courant fourni par la broche **Vin** en **2** est limité par les caractéristiques de la diode **D**. (*Vin est également désignée par **UTN** sur certains schémas électroniques d'Arduino.*) De plus, les parasites générés par les moteurs électriques peuvent transiter vers Arduino via le régulateur intégré fournissant le +5Vcc au microcontrôleur. Cette solution n'est donc acceptable que si les moteurs utilisés sont de faible puissance et ne consomment qu'un courant modéré. Il est préférable d'alimenter Arduino soit par sa ligne USB soit par une alimentation externe, et utiliser pour l'interface de motorisation une alimentation à part branchée sur le connecteur **4**. Dans ce cas il faut retirer le cavalier **3** de son support. Noter que les servomoteurs sont reliés directement au +5Vcc d'Arduino par la broche **5**. On ne peut utiliser les connecteurs dédiés **6** que si les modèles utilisés restent dans des fourchettes de consommation faibles. Dès qu'ils deviennent plus exigeants en courant électriques, il faut comme pour les moteurs DC les alimenter par un "bloc" extérieur.

UTILISATION DU CIRCUIT SHIELD AVEC DEUX SERVOMOTEURS.

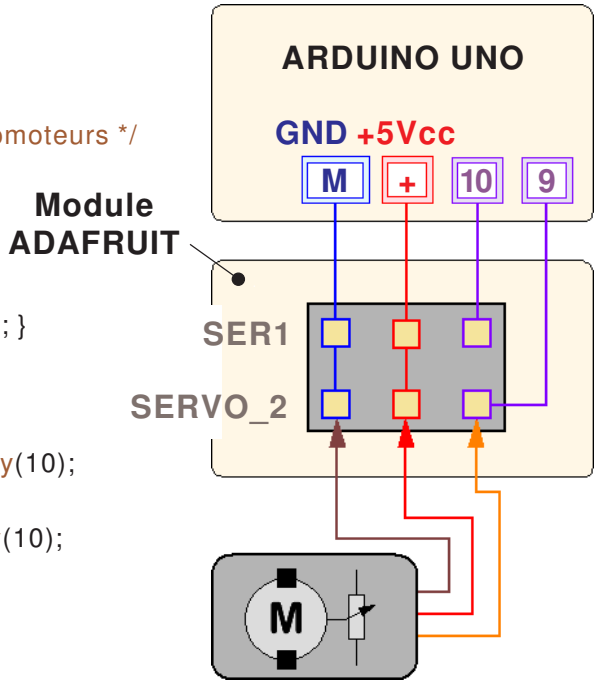
Comme montré sur les schémas Fig.2 et Fig.3, le petit circuit imprimé d'ADAFRUIT n'apporte strictement rien de plus à la carte Arduino en terme d'électronique. Il ne fait que relier les connecteurs **SER1** et **SERVO_2** à la masse, au +5Vcc d'Arduino et aux sorties binaires **B9** et **B10** en respectant l'ordre des fils sur les fiches à trois broches. Comme les servomoteurs puisent leur énergie directement sur le +5Vcc d'Arduino il importe d'en vérifier la consommation. Si elle n'est pas compatible avec les possibilités de la prise USB il faut alimenter les moteurs à part. La programmation est donc banale et utilise les informations données à ce sujet dans les pages 38 et 39.

Fig.3

Exemple de pilotage ANGULAIRE avec la librairie Servo.h :

Programme Test_4_avec_deux_Servomoteurs.ino

```
/* Test de la carte ADAFRUIT pour le pilotage de deux servomoteurs */
#include <Servo.h>
Servo Servo1; Servo Servo2;
void setup() {
  Serial.begin(115200); Servo1.attach(9); Servo2.attach(10); }
int I;
void loop() {
  for (I=0; I<180; I++)  {Servo1.write(I); Servo2.write(I); delay(10);
                          Serial.print(" I = ");Serial.println(I); }
  for (I=180; I!=0; I--) {Servo1.write(I); Servo2.write(I); delay(10);
                          Serial.print(" I = ");Serial.println(I); } }
```



Programmation et utilisation du Shield ADAFRUIT :

Comme c'est les cas presque chaque fois qu'un composant spécifique ou qu'une carte électronique spécialisée sont disponibles sur le marché, une ou des bibliothèques sont développées et mises en ligne pour le public. Le shield ADAFRUIT n'échappe pas à cet adage et plusieurs librairies le concernant sont disponibles sur internet. En particulier AFMotor.h est dédiée à ce module d'interfaçage et couvre déjà pas mal les besoins de la programmation des petites motorisations .

UTILISATION DES MOTEURS À COURANT CONTINU.

Bien que l'électronique autorise le branchement d'un moteur par opérateur de commutation, *(Cas où un seul sens de rotation est suffisant.)* seuls les liaisons en H sont prises en compte par la bibliothèque AFMotor.h, mais avec pour corollaire la possibilité d'imposer à convenance un sens de rotation. On peut dans ces conditions gérer simultanément jusqu'à quatre petits moteurs à courant continu.
Le programme Test_2_moteurs_DC_avec_rampe_de_vitesse.ino est un exemple de mise en œuvre de la carte électronique qui fait tourner quatre moteurs *(Le maximum possible)* dans un seul sens pour simplifier le programme, mais avec une rampe de variation de vitesse en accélération et ralentissement.
Le programme Test_3_moteur_DC_avec_Rampe_et_Repos.ino est une variante du précédent qui ne fait tourner qu'un seul moteur sur le port #4, mais dans les deux sens, toujours avec une rampe de variation de vitesse en accélération et ralentissement. Dans ce programme la déclaration le 1kHZ par défaut est imposée pour la fréquence PWM. Après un cycle dans les deux sens de rotation, le moteur est mis au repos durant une seconde. L'utilisation des procédures fournies par AFMotor.h est quasiment évidente, par contre il faut tenir compte des courants consommés comme vu dans les pages précédentes.

NOTE : Lors des divers essais effectués, aucun de ces moteurs n'était pourvu de condensateurs d'antiparasitage comme montré sur le dessin situé en bas de la page 63. Que ce soit en alimentation directe par Arduino ou par le truchement d'une alimentation extérieure, aucune perturbation n'a été constatée lors du déroulement des programmes. Quand les moteurs utilisés sont de faible puissance et alimentés en tensions faibles, les parasites restent donc suffisamment modérés pour ne pas imposer de condensateurs.

Caractéristiques des petits moteurs utilisés en local.

Plusieurs petits moteurs de récupération sont disponibles "dans les tiroirs".

Moteur DC miniature **RF-300C-09550** :
- Appel de courant au démarrage sous 7Vcc : Inférieur à 100 mA.
- Courant moyen en régime continu à vide sous 7Vcc : 30 mA.
- Courant sous 5Vcc **rotor bloqué** mécaniquement : 800 mA.

Moteur DC miniature **MDN3BL3DRB** :
- Appel de courant au démarrage sous 7Vcc : Environ 200 mA.
- Courant moyen en régime continu à vide sous 7Vcc : 35 mA.
- Courant sous 5Vcc **rotor bloqué** mécaniquement : 800 mA.

Moteur DC MITSUMI **P/NC8974-60010** :
- Pas d'appel de courant au démarrage sous 5Vcc.
- Courant moyen en régime continu à vide sous 5Vcc : 150 mA.
- Courant sous 5Vcc **rotor bloqué** mécaniquement : 500 mA.

Moteur DC "moyen" **HC385MG** :
- Pas d'appel de courant au démarrage sous 5Vcc.
- Courant moyen en régime continu à vide sous 5Vcc : 90 mA.
- Courant sous 5Vcc **rotor bloqué** mécaniquement : 700 mA.

Moteur DC "moyen" sans référence :
- Appel de courant au démarrage sous 5Vcc : Environ 80 mA.
- Courant moyen en régime continu à vide sous 5Vcc : 50 mA.
- Courant sous 5Vcc **rotor bloqué** mécaniquement : 700 mA à 800 mA.

UTILISATION DES MOTEURS PAS À PAS.

Contrairement aux caractéristiques des moteurs à courant continu qui n'influencent pas réellement la programmation, le codage pour les moteurs PAS À PAS est directement fonction de leur résolution. La procédure setSpeed(RPM) qui gère la vitesse de rotation utilise la résolution du moteur utilisé. Outre les problèmes liés au courant consommé, il faudra tenir compte du nombre de **pas par tour**, raison pour laquelle l'encadré donné en bas de cette page précise les caractéristiques des moteurs utilisés en local. On peut noter au passage qu'un moteur PAS À PAS tel que le 28BYJ-48 n'est pas utilisable car le point milieu des deux bobinages est commun et perturberait la commutation en H.

UTILISATION ÉLÉMENTAIRE DES MOTEURS PAS À PAS.

Par élémentaire il faut traduire : Pas de pilotage simultané de deux moteurs PAS À PAS et pas de rampes d'accélération ou de ralentissement. En effet, la bibliothèque AFMotor.h ne prévoit pas de procédure pour traiter les rampes d'accélération ou de ralentissement. Par ailleurs le pilotage simultané de deux moteurs pas à pas est laissé à l'initiative du programmeur par usage de la procédure onestep.

Exemple de pilotage avec les quatre modes possibles :

Le programme Test_5_avec_Moteur_pas_a_pas_et_4_modes.ino assure le pilotage d'un moteur PAS À PAS avec une boucle qui enchaîne les divers modes SINGLE, DOUBLE, INTERLEAVE et MICROSTEP avec une pause release d'une seconde entre chaque sens de rotation au cours desquels le moteur effectue un tour complet. Le programme est écrit pour des STP-42D144 à 200 pas par tour. On peut ainsi expérimenter les comportements qui en résultent. La vitesse programmée est de 15 tours par minute. Comme on peut l'observer dans le listage partiel du programme donné ci-dessous, le nombre de pas et le mode de commutation sont passés en paramètre dans une procédure gérant un cycle complet.

```
void Faire_un_aller_retour(int NB_PAS, byte MODE) {
  Mon_pas_a_pas.step(NB_PAS, FORWARD, MODE);
  Mon_pas_a_pas.release(); delay(1000);
  Mon_pas_a_pas.step(NB_PAS, BACKWARD, MODE);
  Mon_pas_a_pas.release(); delay(1000);}
```

Exemples d'appels :

```
Faire_un_aller_retour(200, SINGLE);
Faire_un_aller_retour(200, DOUBLE);
Faire_un_aller_retour(400, INTERLEAVE);
Faire_un_aller_retour(200, MICROSTEP);
```

Caractéristiques des moteurs PAS À PAS utilisés en local.

Trois petits moteurs pas à pas de récupération sont disponibles "dans les tiroirs".

Moteurs **STP-42D144** et **17PM-K212-P1T** :

Ces deux moteurs sont strictement équivalents.
Tension nominale : 7Vcc.
Courant : 0,7 A.
Impédance : Deux bobinages de 10Ω. *(Sorties sur 4 fils.)*
Couple : 3,2 cm*kg.
Résolution : 200 pas par tour.
Angle : 1,8° par pas.
Poids : 175 grammes.
Mesures effectuées lorsque des résistances de 10Ω sont ajoutées en séries avec chaque bobinage : 200mA par bobinage sous 5Vcc et 300mA sous 7Vcc.

Moteur **SANYO 103-490-0121** :

Deux bobinages à point milieu.
Résolution : 400 pas par tour.
Angle : 0.9° par pas.
Impédance : 34Ω par bobine élémentaire.
Courant par bobinage :
75mA sous 5Vcc, 100mA sous 7Vcc.
Courant par bobine élémentaire :
147mA sous 5Vcc, 200mA sous 7Vcc.

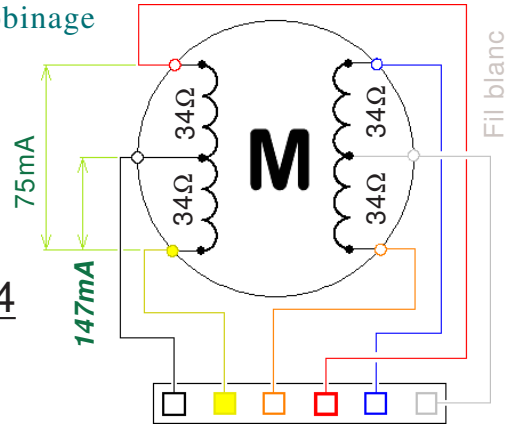


Fig.4

NOTE : Si le sens de rotation du moteur pas à pas à deux bobinages séparés n'est pas celui souhaité, il suffit de permuter les deux fils le branchement d'un bobinage pour l'inverser.

Exemple de pilotage simultané de deux moteurs PAS À PAS :

Le programme Test_6_Deux_pas_a_pas_simultanes réalise le pilotage simultané de deux moteurs pas à pas dans une boucle qui fait appel à la procédure dédiée onestep d'AFMotor.h.
Ce programme est simplifié car les deux moteurs sont de même résolution et effectuent une rotation d'amplitude égale. Mais il reste facile de transposer ce logiciel de base à des moteurs de résolutions différentes ou si la rotation angulaire désirée est différente sur chaque unité.

```
#include <AFMotor.h>

const int DELAI = 1; // Délai effectué entre chaque pas.

AF_Stepper Moteur_1(200, 1); // Moteur 200 pas/tour sur le port #1.
AF_Stepper Moteur_2(200, 2); // Moteur 200 pas/tour sur le port #2.

void setup() { }

void loop() {
  Faire_un_tour_sur_les_deux_moteurs(FORWARD, SINGLE);
  Faire_un_tour_sur_les_deux_moteurs(BACKWARD, SINGLE);
  Faire_un_tour_sur_les_deux_moteurs(FORWARD, DOUBLE);
  Faire_un_tour_sur_les_deux_moteurs(BACKWARD, DOUBLE);
  Faire_un_tour_sur_les_deux_moteurs(FORWARD, INTERLEAVE);
  Faire_un_tour_sur_les_deux_moteurs(BACKWARD, INTERLEAVE); }

void Faire_un_tour_sur_les_deux_moteurs(boolean SENS, byte MODE) {
   int MAX=401; if (MODE=INTERLEAVE) {MAX=801;}
   for (int NB_PAS = 1; NB_PAS < MAX; NB_PAS++)
    {Moteur_1.onestep(SENS, MODE); Moteur_2.onestep(SENS, MODE);
     delay(DELAI);}
   Moteur_1.release(); Moteur_2.release(); delay(1000);}
```

Ce listage est partiel, car le logiciel réel intègre des instructions d'affichage de l'état de déroulement du programme sur la ligne série USB.

Noter que l'instruction onestep() ne permet pas d'utiliser le mode ~~MICROSTEP~~ et **la vitesse n'est plus gérée** par la procédure setSpeed(RPM). Il faut donc imposer un petit délai entre chaque pas pour gérer la vitesse. Dans ce programme la constante DELAI permet de tester la rapidité maximale permise sur les moteurs PAS À PAS connectés à l'interface de puissance d'ADAFRUIT. On peut observer que le MODE est passé en paramètre dans la procédure qui gère la rotation des moteurs, mais également le sens de rotation par le truchement d'un booléen.

REMARQUE : Sur Internet on rencontre souvent des programmes qui remplacent la procédure onestep d'AFMotor.h par l'instructions step :

```
PAS_A_PAS.step(1, FORWARD, SINGLE);
PAS_A_PAS.onestep(FORWARD, SINGLE);
```

} *Instructions équivalentes.*

Ces deux instructions conduisent au même résultat, mais préférer onestep() qui est plus lisible et traduit plus directement sa finalité dans le code du logiciel.

MIXAGE DES TROIS TYPES DE MOTORISATION POSSIBLES.

Comme précisé sur la fiche dédiée au SHIELD d'ADAFRUIT on peut à convenance piloter plusieurs types de moteurs simultanément, la combinatoire étant fonction des modèles connectés. La bibliothèque AFMotor.h ne permet pas de brancher des charges "à sens de rotation unique" comme montré sur la Fig.4 de la fiche relative au Pont en H et circuit intégré L293D. Toutefois, sans faire appel à des programmes plus élaborés, il est déjà possible d'agencer des combinaisons très étoffées. Le programme Test_7_Trois_types_de_M_simultanes.ino en est un exemple dans lequel tous les opérateurs en H du module électronique sont mis à contribution. Un seul SERVOMOTEUR a été intégré à ce petit projet pour alléger "la lisibilité" du programme, mais naturellement en brancher un deuxième reste élémentaire.
ATTENTION : Une alimentation EXTérieure est fortement conseillée sur +VS vu le nombre de moteurs.