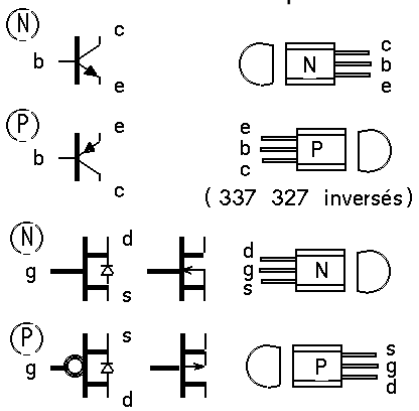
http://www.didel.com/ info@didel.com

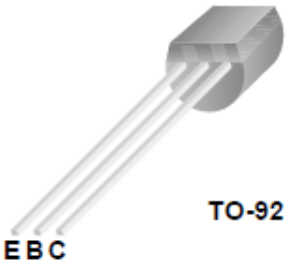
www.didel.com/diduino/Composants04.pdf

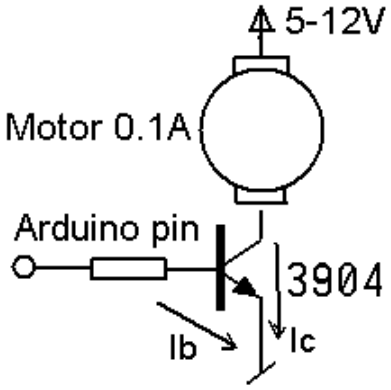
Composants électroniques avec le Diduino
Partie 04 – Transistors, Moteurs, PWM

C14 Transistors et ponts

Dans les grandes lignes, il y a 4 types de transistors, bipolaires et MOS, de type N et P. Avec leurs variantes de puissance. Le 2N3904 est de type N et il est utilisé le plus souvent comme amplificateur inverseur. Le courant de sortie d'une pin d'un microprocesseur est limité à 20mA, et ne peut commander qu'un buzzer ou un moteur de plus de 50 Ohm. Avec un transistor, le courant de la base vers l'émetteur est amplifié. Un courant ~100 fois plus grand va alors du collecteur vers l'émetteur.





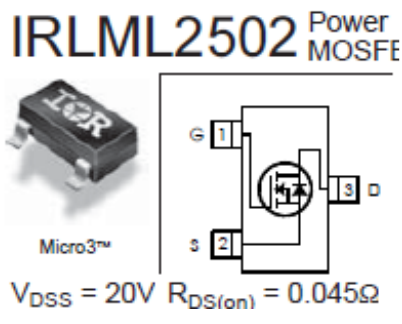


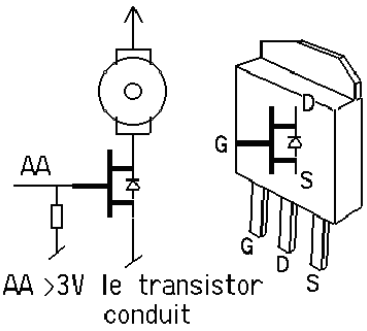
Comment trouver cette résistance interne de 6 Ohm? Dans les spécifications, faciles à trouver sur Internet, il faut repérer la tension de saturation, 0.3V pour un courant de 50mA, donc une résistance interne de 0.3V / 0.05A = 6 Ohm.

DC Current Gain	I_C = 10mA, V_{CE} = 1.0V	100
Collector-Emitter Saturation Voltage	I_C = 50mA, I_B = 5.0mA	0.3 V

0.3V 50mA Ri=0.3/0.05 = 6 Ohm

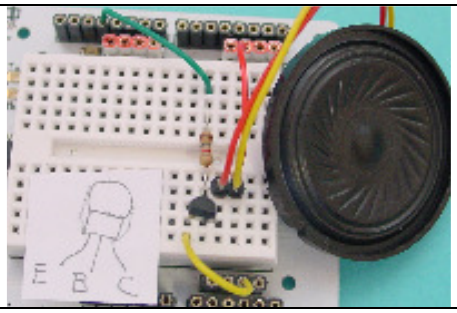
Un transistor MOS n'a pas besoin de résistance en série sur la base (appelée Gate). La résistance interne de quelques centièmes d'Ohm permet de commander des courants de 1 Ampère même avec un transistor miniature, puisque la dissipation de température est faible.





Si un haut-parleur avec une résistance de 8 Ohm est utilisé, un transistor est nécessaire et le son sera nettement plus fort et de meilleure qualité a cause de la dimension du haut-parleur. Des amplis audios à 6 pattes (LM386) sont préférables à un simple transistor.

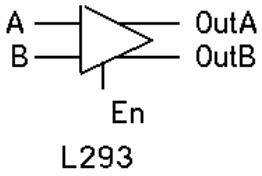
Attention, quand on a fini de faire un son, il faut éviter que du courant passe dans la bobine, ce serait du courant transformé en chaleur inutilement! Et alimenté par pile, la pile sera vidée très rapidement.



Pour commander un moteur et le faire tourner dans les deux sens, il faut 4 ou 6 transistors amplificateurs. Le circuit L293 en boîtier DIL16 est bien connu des Arduinophiles, qui ne savent pas que la résistance interne du circuit est de 2.6 Ohm. A 5V, avec un moteur jouet de 3 Ohm, la moitié du courant est perdue, mais le moteur tourne! A 12V, l'efficacité est acceptable.

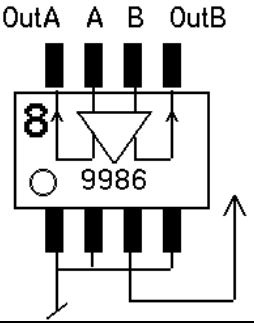
En	A	B	OutA	OutB
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0
0	X	X	-	-

X 0 or 1
- open circuit



A	B	OutA	OutB
0	0	0	0
0	1	0	1
1	0	1	0
1	1	-	-

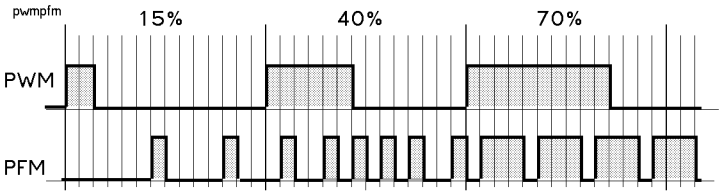
- open circuit



Le circuit Si9986 a un boîtier so8 plus petit et accepte une tension de 3 à 6V. Sa résistance interne est de 1.2 Ohm. Sa table de vérité n'est pas compatible avec celle du L293 et peut poser problème avec les routines PWM d'Arduino (voir plus loin).
Il existe naturellement tout un catalogue de shields et de drivers pour des moteurs plus puissants.
Quelle est la différence entre les sorties à 0 0 (état imposé à 0V) et à - - (ouvertes). Faites tourner l'axe d'un moteur à la main. Vous le lancez et il fait plusieurs tours. Court-circuitez les sorties, il s'arrête rapidement à cause du courant qui s'établit dans les bobines et génère une force contre-électromotrice. Pour un driver de moteur, quand les deux sorties sont à zéro, cela correspond à un mauvais court-circuit qui freine le moteur. Si les transistors de sortie sont inactifs (Enable = 0 du 293), alors le moteur est libre.

C15 Commande proportionnelle

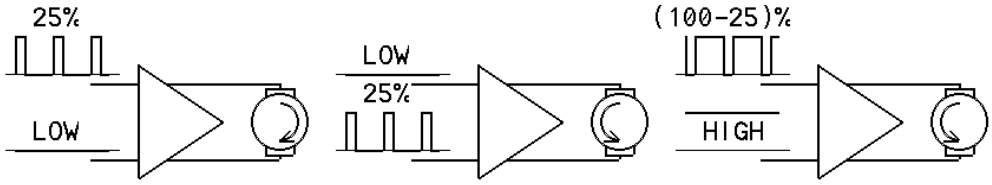
Le PWM (pulse width modulation) travaille avec une fréquence fixe et génère des impulsions de largeur variable.



Le PFM (pulse frequency modulation) travaille avec des impulsions de largeur fixe, positives jusqu'à 50%, puis négatives.
www.didel.com/kidules/KiPfm.pdf
Les deux documents cités expliquent comment programmer le Pwm et Pfm avec des routines appelées par interruption. Les processeurs ont une circuiterie qui permet de configurer des registres et compteurs internes pour générer du PWM sur certaines pins. Arduino utilise des routines par interruption sur d'autres pins.
Pour les cartes utilisant un AtMega168 ou 328, un PWM 8 bits est créé sur les pins 3, 5, 6, 9, 10, et 11 avec l'instruction `analogWrite (pin, pourcentPWM) ;` Le % PWM est une valeur entre 0 et 255, mais il ne faut pas trop compter sur la précision. Le terme de "analog" est absurde, la sortie étant digitale pulsée à 500 Hz environ.
Testons l'éclairage proportionnel d'une Led. Le signal Pot d'un potentiomètre est câblé sur la pin16 par exemple, déclarée en entrée. Une Led est câblée sur la pin8 par exemple. Après déclaration de la variable val et set-up, le programme est:

```
void loop()
{
  val = analogRead(Pot);    // valeur lue 0 à 1023
  analogWrite(Led, val/4); //on divise par 4, car la valeur PWM est entre  0 to 255
}
```

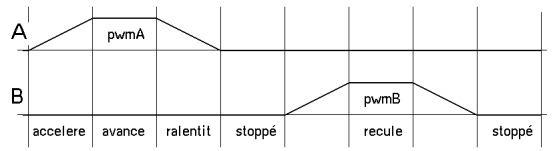
Pour commander un moteur bidirectionnel, on applique le signal PWM d'un côté ou de l'autre de l'ampli. S'il n'y a que l'un des côtés qui peut être piloté en PWM, pour changer de sens on active la sortie fixe et on complémente la valeur du PWM sur l'autre sortie.



Cette deuxième solution fait apparaître une différence entre le 293 et le 9986. Avec le 9986, l'état HIGH sur les deux entrées est roue libre. Donc, avec notre exemple, le moteur va tourner dans un sens en alternant 25% de puissance et 75% blocage, et dans l'autre sens 25% de puissance et 75% de roue libre. La vitesse ne sera pas la même!.

Teston un premier programme qui utilise les deux sorties PWM sur les pins 5 et 6. Une diode bicolore avec sa résistance série permettra de visualiser le comportement d'un moteur qu'il faudrait câbler avec un ample.. Quatre procédures sont définies pour faire évoluer la vitesse.

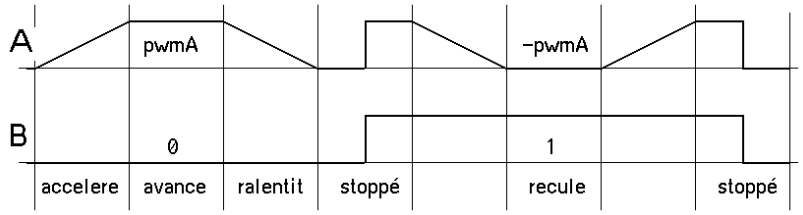
```
//MotAlt1.ino le moteur alterne son sens
// on utilise analogWrite sur les pins 5 et 6
// terster avec une bicolore entre les pins 5 et 6
#define MotAv 5
#define MotRec 6

void setup()
{
  pinMode(MotAv, OUTPUT) ;
  pinMode(MotRec, OUTPUT) ;
}

void accelere (int x)  // psse de 0 à max toutes les x ms
{
  for (int i=0 ; i<256 ; i+=4 )  // 64 pas de 4
  {
    digitalWrite (MotRec,LOW);
    analogWrite (MotAv, i );
    delay (x) ;
  }
}
void ralenti (int x)  // psse de max à 0 toutes les x ms
{
  for (int i=255 ; i>=0 ; i+=-4 )  // 64 pas de 4
  {
    digitalWrite (MotRec,LOW);
    analogWrite (MotAv, i );
    delay (x) ;
  }
}
void accBack (int x)
{
  for (int i=0 ; i<256 ; i+=4 )
  {
    digitalWrite (MotAv,LOW);
    analogWrite  (MotRec, i );
    delay (x) ;
  }
}
```




```
void ralBack (int x)
{
  for (int i=255 ; i>=0 ; i+=-4 )
  {
    digitalWrite (MotAv,LOW);
    analogWrite (MotRec, i );
    delay (x) ;
  }
}
void loop ()
{
  accelere (20) ;
  delay (200) ;
  ralenti (20) ;
  delay (200) ;
  accBack (20) ;
  delay (200) ;
  ralBack (20) ;
  delay (200) ;
}
```

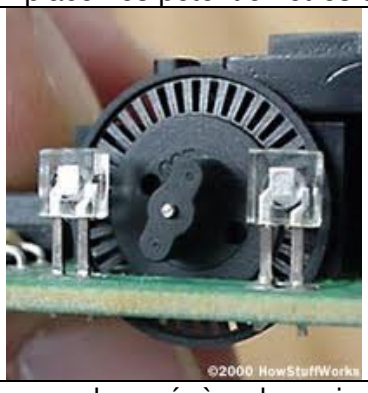
Si on a un seul canal PWM, il suffit de modifier les procédures pour la marche arrière. Les shields moteur qui utilisent les pins 4,5 et 6,7 pour commander les moteurs doivent procéder de cette façon. Le DdRobot également, avec le problème mentionné dû au Si9986.

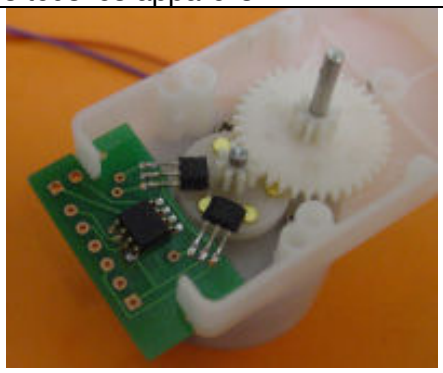
```
// MotAlt2.ino
...
void accBack (int x)
{
  for (int i=255 ; i>=0 ; i+=-4 )
  {
    digitalWrite (MotRec,HIGH);
    analogWrite  (MotAv, i );
    delay (x) ;
  }
}
```



C16 Encodeurs

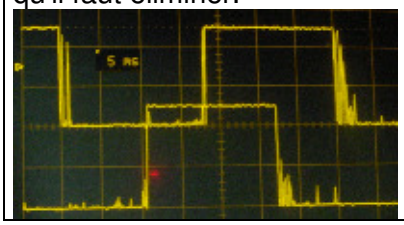
Un encodeur (quadrature encoder) permet de connaître la position angulaire d'un moteur, donc sa vitesse. On le trouve dans toutes les souris "mécaniques", à l'arrière de certains moteurs et pour remplacer les potentiomètres dans tous les appareils..

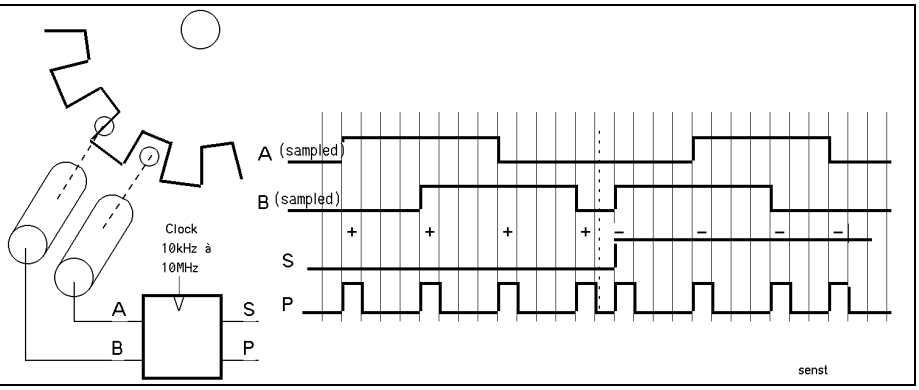




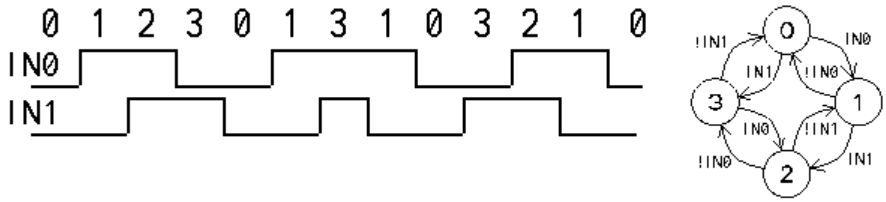
Un encodeur génère deux signaux en quadrature et le déphasage change selon la direction.

Les capteurs peuvent être optiques, magnétiques ou mécanique, avec dans ce cas des rebonds de contact qu'il faut éliminer.





Le décodage se fait avec une machine d'état, les signaux étant échantillonnés suffisamment fréquemment pour ne pas rater de transition.

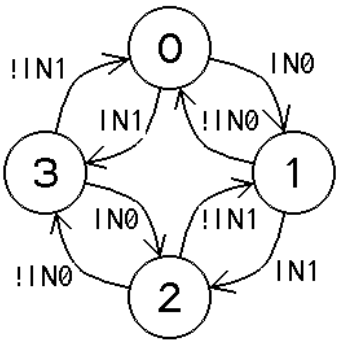


Voir www.didel.com/kidules/KiEnco.pdf pour un exemple de programmations et www.didel.com/mot/RomEnco.pdf pour une approche plus générale
Avec un encodeur mécanique, on retrouve le problème des rebonds de contact, qu'il faut éliminer par échantillonnage.
Arduino propose des "Encoder Library"

La programmation d'un encodeur est un bon exercice pour comprendre ce qu'est une machine d'état et la commande switch..case.
Dans chacun des 4 états on doit tester deux fronts pour déterminer l'état suivant. Le test se fait toutes les 2 millisecondes à cause de rebonds de contact. et le résultat est un compteur qui augmente ou diminue selon le sens de rotation.
Pour le test, un encodeur rotatif a été câblé sur un kidule universel. Les deux sorties ont des Leds en série avec la pull-up et sont câblées sur les pins 2 et 3 appelées e1 et e2 et définies en entrées dans le set-up.
Cette application ne suggère pas de nom évocateur pour les 4 états, donc pas de enum pour nommer les états 0,1,2,3, mémorisés dans la variable `int etat = 0;` (il faut l'initialiser).
le compteur est défini par `int cnt = 0 ;`

Pour faire mieux ressortir la logique de ce programme, compactifions en ligne. Aussi, si la condition est `(val==1)` on peut écrire `(val)` et si la condition est `(val==0)` on peut écrire `(!val)`. Le point d'exclamation signifie inversion. Le passage du diagramme d'état au codage est évident.

```
switch (etat)
{
  case (0) : // IN0=0 IN1=0
    if (digitalRead (IN0)) { etat = 1 ; cnt++ ; affi(); }
    if (digitalRead (IN1)) { etat = 3 ; cnt-- ; affi(); }
    break;
  case (1) : // IN0=1 IN1=0
    if (!digitalRead (IN0)) { etat = 0 ; cnt-- ; affi(); }
    if (digitalRead  (IN1)) { etat = 2 ; cnt++ ; affi(); }
    break;
  case (2) : // IN0=0 IN1=1
    if (digitalRead  (IN0)) { etat = 3 ; cnt++ ; affi(); }
    if (!digitalRead (IN1)) { etat = 1 ; cnt-- ; affi(); }
    break;
  case (3) : // IN0=1 IN1=1
    if (!digitalRead (IN1)) { etat = 2 ; cnt-- ; affi(); }
    if (!digitalRead (IN1)) { etat = 0 ; cnt++ ; affi(); }
    break;
}
```



L'affichage du compteur se fait au terminal, à chaque transition par la fonction `affi()`. Si on tourne trop vite, la transmission série des 5 caractères va faire perdre des transitions. Le programme complet est sous Encodeur.ino

Suite www.didel.com/diduino/Composants05.pdf