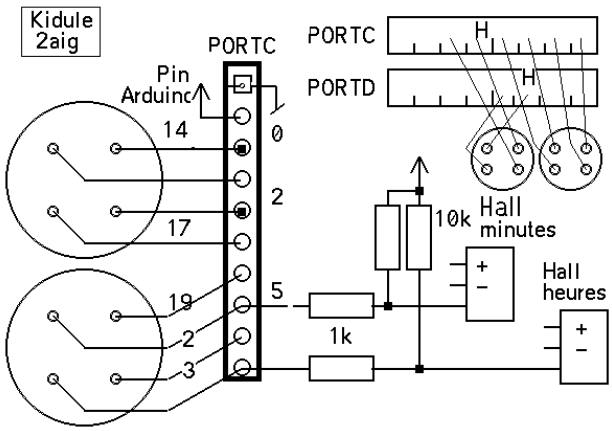
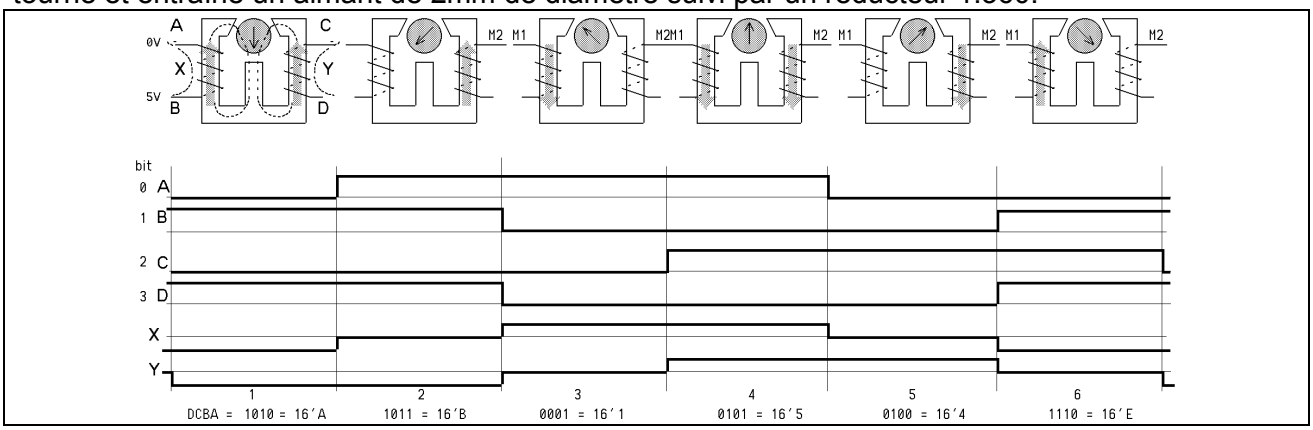
Deux moteurs pas-à-pas sur le même axe

Le moteur double Vid28.05 (Google "Vid28") est utilisé en instrumentation (tableau de bord des voitures). Ce n'est pas un moteur à 4 phases, comme les moteurs documentés sous Arduino, mais un moteur Lavet à 6 phases. Les bobines ont une résistance de 280 Ohm, on peut donc les connecter directement par des sorties d'un microcontrôleur. Les axes sont entraînés via un réducteur 1:180.
Le moteur du haut commande la grande aiguille des minutes.

Des capteurs de hall ont été ajoutés avec résistance de protection. connecteur broche 9 pour les heures, 7 pour les minutes.
Cet interface montre la nécessité d'être attentif aux noms des signaux avant de commencer à programmer:
Numéros des broches du connecteur 1 à 10
Bits du port Kidules 0 à 7
Pins Arduino 14-19 et 2-3
Ports AVR 328 PORTC bits 0-5 PORTD bits 2-3



Séquence à 3 phases, 2 bobines séparées
La séquence des phases est donnée ci-dessous. Le champ magnétique créé par des deux bobines tourne et entraîne un aimant de 2mm de diamètre suivi par un réducteur 1:360.



Pour le moteur connecté sur les 4 bits de poids faible du PORTC, la séquence de pas reprend directement les valeurs binaires du graphique ci-dessus. Elle se code dans un tableau de 6 valeurs.

```
byte step[6]={0x0A,0x0B,0x01,0x05,0x04,0x0E};
```

que l'on parcourt avec un compteur par 6, avec un `if` pour recommencer

```
i++; if(i==6) i=0; // compte 0 1 2 3 4 5 0 1 2 ....
```

Pour changer de sens, il faut compter dans l'autre sens

```
i--; if(i==0) i=6; // décompte 6 5 4 3 2 1 0 6 Il faut corriger l'index en demandant etat[i-1]
```

Les programmes se trouvent dans le dossier par défaut www.didel.com/kidules/CKiPas2Aig.zip

Le programme `CKiPa1.ino` fait tourner l'aiguille des heures. Le moteur est câblé sur les 4 bits de poids faible du PORTC.
Il suffit de l'initialiser en sortie et d'écrire directement sur ce port. Entre chaque pas, il faut naturellement une attente qui fixe la vitesse de rotation. La fréquence maximale des pas est supérieure à 1 kHz, les délais sont en microsecondes.

```
//CKiPa1.ino
byte avance[6]={0x0A,0x0B,0x01,0x05,0x04,0x0E};
void setup()
{
  DDRC = 0b11111111 ; //direction
  PORTC = 0 ;  //moteur non excité
}

void loop()
{
  for (int i=0; i<6 ; i++ )  {
    PORTC = avance[i];
    delayMicroseconds(5000); // max vit 3000? à 5V
  }
}
```

Pour agir sur le moteurs des minutes, il faut remarquer qu'il a 2 bits sur le portC et 2 bits sur le PortD. Il faut initialiser ces 2 ports, en ne modifiant pour le portD que les bits concernées. On utilise la même table et on envoie pour ce premier test 2 bits sur C et 2 bits sur D.

```
//CKiPa1m.ino  Avance l'aiguille des minutes
byte avance[6]=
{0x0A,0x0B,0x01,0x05,0x04,0x0E};
void setup()  {
  DDRC  = 0b11111111 ;
  PORTC = 0 ;
  DDRD |= 0b00001100 ;
  PORTD = 0 ;
}
```

```
void loop() {
  for (int i=0; i<6 ; i++ )  {
    PORTC = (avance[i] << 4) & 0x30 ;
 // 00ba0000
    PORTD = avance[i] & 0x0C ;
 // 0000dc00
    delayMicroseconds(5000);
  }
}
```

On voit que avec cette table, l'aiguille des minutes tourne en sens inverse. On peut changer la table, ou permuter deux bits sur une des deux bobines, ce qui aurait été facile à faire sur le circuit imprimé, mais est plus délicat par programmation (<4 instructions supplémentaires!).
La solution est comme on a vu, de parcourir la table à l'envers.
Exercice: faire tourner les 2 moteurs en même temps dans le même sens.

Pour accélérer le moteur, on joue avec une 2e boucle for et on remarque que le moteur peut tourner à une vitesse supérieure à sa vitesse de démarrage.

A chaque pas, on modifie la vitesse. et un while (1) permet de rester à la vitesse maximum. La Led Arduino sur la pin 13 signale cet état.

Si on bloque l'aiguille, elle ne repart pas au-delà d'une certaine vitesse. Mais on peut la relancer.

Le bouton reset permet de relancer le programme sans le télécharger.

```
//CKiPa2.ino accélère jusqu'à un vitesse max
// La Led13 est allumée si cette vitesse max est atteinte
#define Led 13
#define LedOn digitalWrite (Led,HIGH)
#define LedOff digitalW
byte etat[6]={0x0A,0x0B,0x01,0x05,0x04,0x0E};
void setup() {
  DDRC = 0b11111111 ; //direction
  PORTC = 0 ;  //moteur non excité
}
int j ;
#define PeriodeMin 700
void loop()  {
  for (j= 1000 ; j > PeriodeMin ; j-- ) {
     for (int i=0; i<6 ; i++ ) {
       PORTC = etat[i];
       delayMicroseconds(j);
     }
  }
  while (1) {  // continue à la fréquence max
    for (int i=0; i<6 ; i++ ) {
      PORTC = etat[i];
      delayMicroseconds(j);
    }
  }
}
```

On a agi sur la vitesse. Il faut aussi savoir agir sur l'angle, c'est-à-dire sur le nombre de pas.

```
//CKiPa3.ino avance-recule de 100 pas
byte Avance[6]=
{0x0A,0x0B,0x01,0x05,0x04,0x0E};
```

(suite 1ere colonne)

```
void loop()  {
  // on fait 100 pas dans un sens
  for (int j=0; j<100;j++) {
    for (int i=0; i<6; i++)  {
      PORTC = etat[i] ;
      delayMicroseconds(2000);
    }
```

```
void setup() {
  DDRC = 0b11111111 ; //direction
  PORTC = 0 ;  //moteur non excité
}
```

```
  }
  // on fait 100 pas en sens inverse
  for (int j=0; j<100;j++) {
    for (int i=5; i>=0; i--)  {
      PORTC = etat[i] ;
      delayMicroseconds(2000);
    }
  }
}
```

Exercice: Aller-retour en modifiant la vitesse.

Bouger les deux moteurs

Pour commander les deux moteurs, on a vu qu'il faut mélanger des bits de deux ports du processeur, ce qui se fait avec la fonction `WriteKi();` qui permet de raisonner sur un mot de 8 bits.

```
void WriteKi (byte kk)
{
  PORTC  = kk & 0x3F ;       // copie les 6 bits
  PORTD &= (kk>>4) | ~0x0C; // copie les zéros
  PORTD |= (kk>>4) &  0x0C; //copie les uns
}
```

Explications sur le port Kidule: www.didel.com/diduino/PortK.pdf

Le programme `CKiPa4.pde` fait tourner les deux moteurs à la même vitesse. En changeant la table, ils tournent dans le même sens ou en sens inverse.

```
//CKiPa4.ino
. . . définitions, set-up, fonction WriteKi
// les deux moteurs tournent en sens contraire
byte etat2m[6]={0x77,0x66,0xEE,0x88,0x99,0x11};

void loop()
{
  for (int i=0; i<6; i++)
  {
    WriteKi (etat2m[i]) ;
    delayMicroseconds(2000);
  }
}
```

Evidemment, on veut pouvoir commander les deux moteurs indépendamment. L'un après l'autre est facile. On fait un certain nombre de pas avec un moteur, puis avec l'autre en copiant des éléments de Pas3.ino avec une table séparée pour les deux moteurs, et en écrivant sur la bonne moitié du port. L'utilisation de la routine bloquante delay() est limitatif. Par interruption on utilise un timers. Par programmation, on décide toutes les 500 microsecondes si un pas doit se faire pour chaque moteur (voir www.didel.com/kidules/CKiDelta.pdf)

Lecture des Hall

On doit brièvement couper l'excitation moteur pour lire les capteurs de Hall, qui donnent un état 0 lorsque le champ magnétique est présent. C'est inutile de lire trop souvent, tous les 6 pas correspond à un angle de 2 degrés, selon la proximité de l'aimant, le Hall est actif pour 3 à 8 degrés de déplacement de l'aiguille.

Pour le programme de test, on a défini les deux actions qui définissent la direction des bits sur lesquels les Hall sont lus. Un délai de 500 us est nécessaire pour que le signal du Hall se stabilise.
(voir www.didel.com/kidules/CKiClock.pdf)

```
//CKiPa5.ino  Test des 2 capteurs deHall
. . . .  définitions, set-up
void loop () {
  for (int i=0; i<6 ; i++ )  {
    KiWrite ( etat[i] | (etat[i]<<4) );
    delayMicroseconds(Periode);
  }
  ModeHall;
  delayMicroseconds (500); // temps commut
  if (MinuteTop) { LedOn;
     delay (20);  // stoppe moteur
  }
  if (HeureTop) { LedOn;
     delay (20);  // stoppe moteur
  }
  ModePas;
  LedOff;
}
```

Exercices: Déplacements décoratifs ou surprenants.
Utiliser un capteur magnétique (boussole) pour montrer le sens du champ.