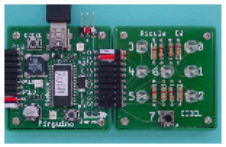
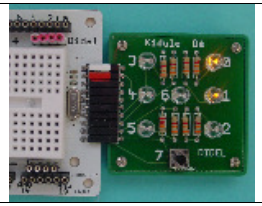
http://www.didel.com/ **info@didel.com**

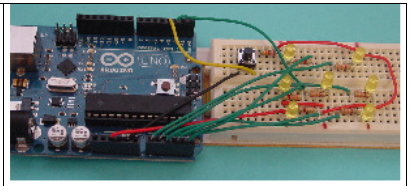
www.didel.com/diduino/KiDe.pdf

Diduino - expériences avec le Kidule Dé

Matériel nécessaire
Vous devez avoir le Kidule-Dé ou un montage équivalent avec un poussoir (actif à 0) et 7 LEDs (actives à 0). Les programmes sont compatibles avec le Diduino et le DdRobot.
Ce document est l'adaptation pour Arduino de www.didel.com/kidules/KiPiDe.pdf qui utilise un Kidule 2550 Pinguino, ou, avec des adaptations mineures une carte KiCar ou PicStar.
A noter que la doc Pinguino commence avec des exercices avec le Dé. Avec le Diduino, le lecteur a déjà passé quelques heures à jouer (Jouer01.pdf, Cours0x.pdf) et il connaît les instructions simples.

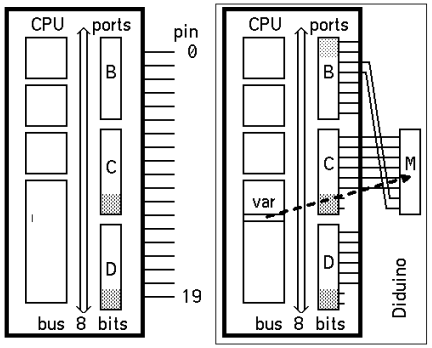

Pinguino KiPiDe



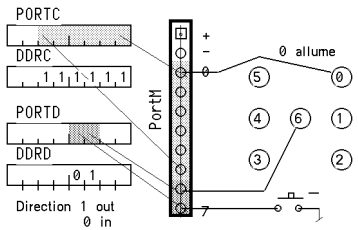


Le processeur

Arduino cache l'intérieur du processeur et ne montre que ses broches (pins) qui sont liées aux ports, parfois incomplets ou avec des pins réservées. Ce qui nous intéresse c'est de travailler avec des mots de 8 bits que l'on "usine" en mémoire, et que l'on transfère dans un port 8 bits. Comprendre un peu cette structure interne d'un processeur est essentiel si on veut maîtriser les applications.
Arduino n'ayant pas de port 8 bits complet utilisable, un port compatible Microdule, le PortM, a été ajouté. La configuration de ce PortM (dire quelles lignes sont en entrées et sorties) suppose des connaissances non essentielles si le but est d'apprendre à programmer. Arduino a caché ces définitions, les nôtres seront encore visibles au début de nos programmes.



Structure des programmes
Un programme doit dire au processeur ce qui est entrée et sortie, c'est le ''set-up'' qui avec Arduino définit des bits. Ici on va travailler directement avec les registres du processeur, comme on le fait dans toutes les applications industrielles du C.
Si nous nous intéressons à allumer des Leds, elles peuvent être vues individuellement comme des pins Arduino. C'est souvent plus efficace de les voir comme un "vecteur", un mot de 8 bits que l'on retrouve sur le connecteur Microdule appelé PortM. Les Leds sont numérotées de 0 à 7 dans l'ordre des bits du PortM. Le tableau ci-contre donne la correspondance, et dans notre premier programme qui clignote la Led du centre, on met en évidence nos deux approches:



Arduino – on utilise un digitalWrite (LedCentre,LOW); pour allumer
C – on place sur le portM le mot binaire 0b10111111 qui active le bit 6, actif à zéro.

Comparons les deux approches. en clignotant la Led du centre

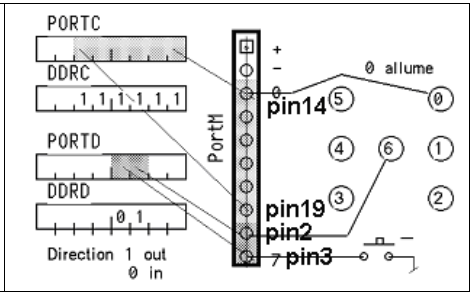
Arduino
```
//CliCentreA.ino
#define LedCentre 2


void setup()
{
  pinMode(LedCentre,OUTPUT);
}




void loop()
{
  digitalWrite (LedCentre, LOW) ;
  delay (500);
  digitalWrite (LedCentre, HIGH) ;
  delay (500);
}
```

C
```
//CliCentreC.ino
#define LedCentreOn  0b10111111
#define LedCentreOff 0b11111111

void setup ()
{
  DDRD = DDRD & 0b00001000 ;  // poussoir en entrée
  DDRC = 0b00111111;  // Leds 0-5
  PORTC = 0b11111111;  // Leds éteintes
}
void WritePortM (int vx) {
  PORTC = vx;
  PORTD = vx<<4;
}

void loop()
{
  WritePortM (LedCentreOn);
  delay (500);
  WritePortM (LedCentreOff);
  delay (500);
}
```

Les programmes se trouvent sous www.didel.com/diduino/KiDuiDe.zip
La version Arduino est le bien connu Blink. En C if faut dire que cette Led au centre est le bit 6 du mot qui représente les 7 Leds du Dé (le 8e bit est le poussoir).
La définition `#define LedCentreOn 0b10111111` prépare ce byte avec le bit à zéro placé où il faut pour allumer la LED. La fonction `WritePortM (LedCentreOn);` transfère cette valeur dans le kidule Dé, via les Ports C et D du processeurs. Les noms sont explicites, c'est ce qui compte. Nous ne connaissons pas encore assez la structure de l'Atmega328 et les instructions du C pour être à l'aise avec le setup. DDRD, DDRC sont des registres de direction, qui disent quels bits sont en entrée ou en sortie. Il y a beaucoup de registres de configuration dans le processeur, qui lui donnent sa flexibilité. Nous ne connaissons pas encore les opérations sur des mots binaires (signes &, <<) mais cela ne doit pas nous empêcher de progresser en utilisant des définitions et des fonctions qui marchent:

`#define LedCentreOn 0b10111111` envoyé sur le portM allume la Led6, éteint les autres
`WritePortM (etat);` envoie sur le PortM les 8 bits de la variable ou constante etat.

Donc si on veut former un 4 sur le dé, on définit `#define Quatre 0b00101101` et pour afficher ce motif, on écrit dans le programme
`WritePortM (quatre);` ou
`WritePortM (valeur);` si on a calculé avant que `valeur=quatre;`
Ça, c'est du C avec lequel on peut aller très loin, mais il faut préparer son voyage!



Pas convaincu par l'intérêt de travailler avec des mots binaires? Allumons une après l'autre les Leds du dé (un chenillard). Avec Arduino, il faut tout déclarer, dire que 7 pins sont en sorties, les mettre à HIGH pour que l'affichage ne soit pas quelconque au début. Et ensuite, il faut activer chaque ligne, attendre, désactiver, activer la suivante etc. Même un Nul fier d'être Nul se demande si on ne peut pas faire mieux!
En C, on raisonne avec une image du Dé en mémoire et on utilise une procédure pour "balancer" cette image sur le Dé. On définit les bits selon le câblage du Dé, on agit sur les ports pour définir, 8 bits à la fois, la direction (input-output) et l'état initial. Ensuite, puisque qu'il faut activer les bits dans l'ordre où ils sont sur le registre, on décale et on teste la condition "tout décalé" pour recommencer. Mathématique, élégant. Et si l'ordre d'allumage est différent, on verra que l'on prépare un tableau avec les valeurs à afficher.

Arduino
```
//CliSeqA.ino
#define Led0 14
#define Led1 15
#define Led2 16
#define Led3 17
```

C
```
//CliSeqC.ino
void setup ()
{
  DDRD &= 0b11110111 ;  // poussoir en entrée
  DDRD |= 0b00000100 ;  // led centre sortie
```

```
#define Led4 18
#define Led5 19
#define Led6  2  // comme ledCentre
#define LedCentre 2
#define PousDe 3
void setup()
{
  pinMode(Led0,OUTPUT);
  pinMode(Led1,OUTPUT);
  pinMode(Led2,OUTPUT);
  pinMode(Led3,OUTPUT);
  pinMode(Led4,OUTPUT);
  pinMode(Led5,OUTPUT);
  pinMode(LedCentre,OUTPUT);
  digitalWrite (Led0, HIGH) ;
  digitalWrite (Led1, HIGH) ;
  digitalWrite (Led2, HIGH) ;
  digitalWrite (Led3, HIGH) ;
  digitalWrite (Led4, HIGH) ;
  digitalWrite (Led5, HIGH) ;
  digitalWrite (LedCentre, HIGH) ;
}
void loop()
{
  digitalWrite (Led0, LOW) ;
   delay (500);
  digitalWrite (Led0, HIGH) ;
  digitalWrite (Led1, LOW) ;
   delay (500);
  digitalWrite (Led1, HIGH) ;
  digitalWrite (Led2, LOW) ;
   delay (500);
  digitalWrite (Led2, HIGH) ;
  digitalWrite (Led3, LOW) ;
   delay (500);
  digitalWrite (Led3, HIGH) ;
  digitalWrite (Led4, LOW) ;
   delay (500);
  digitalWrite (Led4, HIGH) ;
  digitalWrite (Led5, LOW) ;
   delay (500);
  digitalWrite (Led5, HIGH) ;
  digitalWrite (LedCentre, LOW) ;
   delay (500);
  digitalWrite (LedCentre, HIGH) ;
}
```

```
  DDRC |= 0b00111111;  // Leds 0-5
  PORTD |= 0b00000100;
  PORTC |= 0b00111111;  // Leds éteintes
}
void WritePortM (int mm)
{
  PORTC = mm ;
  PORTD = mm>>4 ;
}
```

```
byte motif = 0b00000001 ;
void loop()
{
  WritePortM (~motif);
  delay (500);
  motif <<= 1 ;

  if (motif == 0b10000000)
  {
     motif=0b00000001 ;
  }
}
```

On raisonne avec un motif dit "logique" dans lequel un 1 allume, et au moment du transfert, on inverse (signe ~) pour être compatible avec la contrainte "physique" du câblage (Led allumée par un zéro).

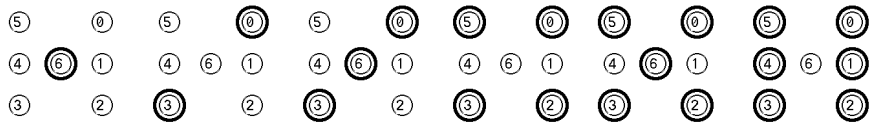
Les opérations logiques sont présentées sous
http://arduino.cc/fr/Main/OperateursBooleens
et mieux expliqueés sous
http://www.commentcamarche.net/contents/c/cop.php3
Un document Cours0x.pdf avec exercices est en préparation.

Exercices: inventez vos chenillards

On a vu que `motif <<= 1 ;` décale le mot binaire à gauche. Des 0 sont injectés à droite.
`motif >>1 ;` décale à droite
Si on veut injecter un 1, on le force après décalage avec une addition: `motif+= 1 ;` ou `motif+= 0b10000000 ;` (plus court d'écrire `motif += 0x80 ;`)
Donc, avec quelques boucles `for` , des délais et des définitions de motifs, on peut créer des jeux lumineux amusants.

Le Dé

Pour un dé électronique, nous avons 6 combinaisons d'allumage à créer. Avec Arduino, il faut chaque fois 7 instructions pour dire quelles Leds on allume et éteint. On connaît leur position sur le connecteur du Kidule Dé, on va donc définir les 6 motifs en binaire.



Le programme `DeFaces.ino` affiche directement ces combinaisons et montre une autre façon de faire des chenillards.


```
//DeFaces.ino
void setup ()
{
  DDRD = 0b00000100;  // led centre sortie
  DDRC = 0b00111111;  // Leds 0-5
  PORTD |= 0b00000100;
  PORTC |= 0b00111111;  // Leds éteintes
}
void WritePortM (int mm)
{
  PORTC = mm ;
  PORTD = mm>>4 ;
}
#define Un 0x40
#define Deux 0x09
#define Trois 0x49
#define Quatre 0x2D
#define Cinq 0x6D
#define Six 0x3F
```

```
void loop()
{
  WritePortM (~Un);
  delay (500);
  WritePortM (~Deux);
  delay (500);
  WritePortM (~Trois);
  delay (500);
  WritePortM (~Quatre);
  delay (500);
  WritePortM (~Cinq);
  delay (500);
  WritePortM (~Six);
  delay (500);
}
```

De là, avec quelques `if` , on a un dé électronique. Il faut un compteur qui compte de 1 à 6 et s'arrête au hasard quand on pèse sur le poussoir pour demander une nouvelle valeur.
Pendant que l'on pèse, on augmente ce compteur spécial (après 6 on a 1) à toute vitesse, donc on ne peut pas deviner quand il faut relâcher pour piper le dé.
Le programme `DeJoue1.pde` est à tester, mais faisons mieux.

Tableaux

Un tableau est une collection de variables qui sont accessibles à l'aide d'un numéro d'ordre (index). Attention, les 6 faces du dé, et leur motif de 0 et 1 qui définit l'allumage, sont numérotés de 0 à 5.
Le tableau se définit `byte Faces[6]={0x40,0x09,0x49,0x2D,0x6D,0x3F);`
La commande `Faces[6]` a rempli 6 positions mémoires successives et on peut retrouver le contenu de la 3e position en écrivant `Faces(2)` (2 et pas 3 car la 1ère est 0).
Pour afficher, c'est très simple puisque l'on peut écrire directement sur le PortM.
Jouez avec le programme `DeJoue2pde` et vérifiez sur 100 lancers que la probabilité d'apparition de chaque chiffre est correcte.
Quand on pèse sur le bouton, on voit toutes les LEDs allumées. En fait elles clignotent ; agitez le Kidule de droite à gauche, en rond, en zig-zag, c'est très joli !
Certaines LEDs du dé s'allument 1/6e du temps, d'autres ½ du temps, etc (vous comprenez pourquoi ?), Donc on voit des segments lumineux de longueur variable. Que se passe-il si on modifie le delay ?

DeJoue1.ino
…

```
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)=LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
  }
  else
  {
    if (counter==1)
     { WritePortM (~Un); }
    if (counter==2)
     { WritePortM (~Deux); }
    if (counter==3)
     { WritePortM (~Trois); }
    if (counter==4)
     { WritePortM (~Quatre); }
    if (counter==5)
     { WritePortM (~Cinq); }
    if (counter==6)
     { WritePortM (~Six); }
  }
}
```

DeJoue2.ino
….

```
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)==LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
  }
  else
  {
   WritePortM (~Faces[counter]);
  }
}
```

Quand on pèse, le compteur avance, mais on ne le voit pas. Quand on lâche, il n'avance plus et on affiche la valeur pointée.

DeJoue3.ino
…

```
byte counter = 1 ;
void loop()
{
  if (digitalRead(PousDe)==LOW)
  {
    counter++;
    if(counter==7)
    {
      counter=1;
    }
    delay (2) ;
    WritePortM (~Faces[counter]);
  }
}
```

Le compteur est toujours affiché. On le voit bouger en agitant le dé. Si on augmente le délai, on peut tricher!