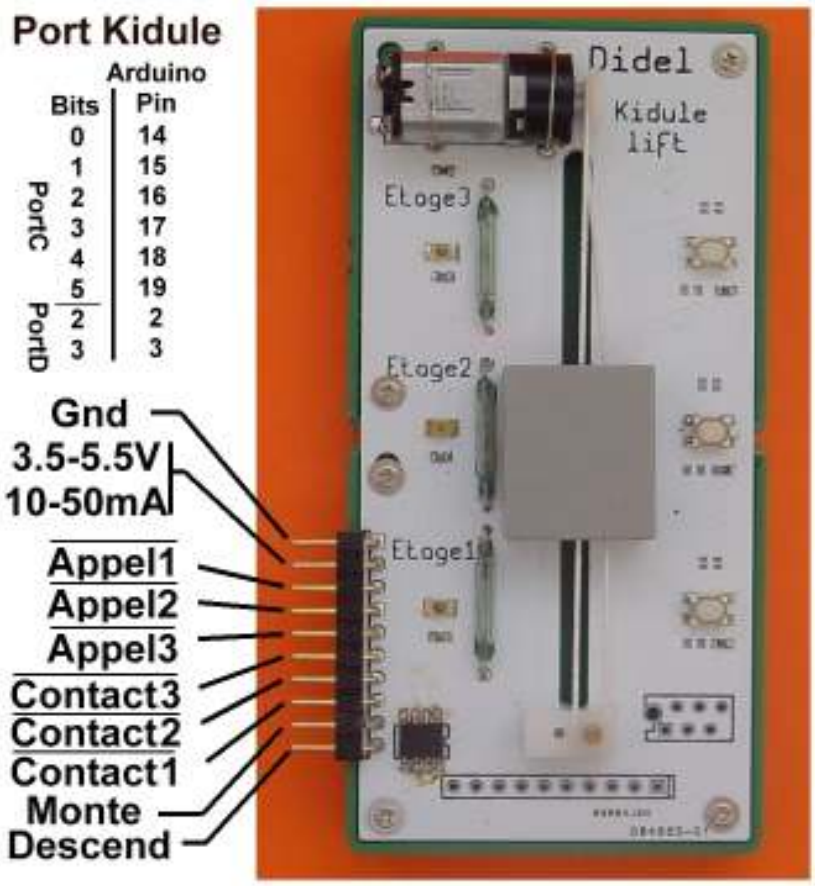
Kidule Ascenseur - logiciel Arduino/C

Ceci suit le cours Cfacile qui utilise les Kidules Feux, Dé et Af4. www.didel.com/kidules/Cfacile.pdf
Il permet de commander un moteur et de bien comprendre l'avantage des machines d'état et la simplicité de leur programmation.

Kidule Ascenseur
L'ascenseur est commandé par un port 8 bits kidule qui comporte un circuit pour commander le moteur, 3 poussoirs (appel d'étage) et 3 Leds (contacts d'étage). La tension est de 3-5V avec un courant max de 0.5A.
Le schéma est simple. Les poussoirs font contact avec le 0V et une résistance vers le 5V donne un état 1 quand le poussoir est relâché. Les Leds sont allumées par un état 0, quand le contact est fermé. On peut aussi les allumer par programme, si le contact est ouvert.



La première chose comme d'habitude est de bien comprendre le câblage et donner des noms clairs aux signaux et aux fonctions. Les poussoirs sont reliés à des pins appelées PinAppel1 2 3. Ces pins sont à l'état 1 au repos. Un poussoir pressé donne un état 0; il faudra en tenir compte en définissant les noms Appel1 2 3 qui sont des noms logiques, Appel1est vrai si on appelle l'ascenseur en pressant sur le bouton.
De même, on a des capteurs d'étage reliées aux PinContact1 2 3, son actifs à 0 et allument la Led associée dans cet état, mais on raisonnera avec Contact1 2 3 actifs.
Il peut être intéressant d'agir directement sur ces Leds PinLed1=PinContact1 etc
Notons encore que la direction n'est pas la même et que si le contact est fermé la Led est allumée de toute façon.
Le processeur ne sait rien de l'application. Ses 8 bits du port Kidule (Ports C et B) sont liés aux lignes de l'ascenseur et il faut associer des noms qui nous sont clairs avec les numéros de bits que le processeur manipule.
Les programmes se trouvent sous www.didel.com/kidules/CKiAsc.zip

Le fichier de définition Arduino est donné ci-dessous.

```
//AscDef.h  Déclarations pour le module Ascenseur
// Cablage - voir figure
#define PinAppel1   14  // bas
#define PinAppel2   15
#define PinAppel3   16 // haut
#define PinContact3 17 //haut
#define PinContact2 18
```

```
// Fonctionnalite ! signaux actifs à zero
#define Appel1   digitalRead (14)  // bas
#define Appel2   digitalRead (15)
#define Appel3   digitalRead (16)  // haut
#define Contact3 digitalRead (17)  //haut
#define Contact2 digitalRead (18)
#define Contact1 digitalRead (19)

#define Monter       digitalWrite (3,1); digitalWrite (2,0)
#define Descendre digitalWrite (3,0); digitalWrite (2,1)
```

```
#define PinContact1 19
#define PinLed3 17 //haut
#define PinLed2 18
#define PinLed1 19
#define PinMotUp 3
#define PinMotDown 2
```

```
#define Stop        digitalWrite (3,0); digitalWrite (2,0)

#define Led3On   digitalWrite (PinLed3,0)
#define Led3Off  digitalWrite (PinLed3,1)
#define Led2On   digitalWrite (PinLed2,0)
#define Led2Off  digitalWrite (PinLed2,1)
#define Led1On   digitalWrite (PinLed1,0)
#define Led1Off  digitalWrite (PinLed1,1)
```

Le fichier de définition en C, qui agit directement sur les bits des registres et des ports est donné pour comparaison, et pour montrer que si l'on regroupe bien les définitions propres au matériel et au compilateur, les programmes sont écrit en pensant à leur fonctionnalité et pas à des contraintes de signaux actifs à 0 ou 1.

```
//AscDefC.h   Kidule Ascenseur  voir schéma pour le câblage
#define Appel1 bitRead (PINC,0)
#define Appel2  bitRead (PINC,1)
#define Appel3  bitRead (PINC,2)
#define Contact3  bitRead (PINC,3)
#define Contact2  bitRead (PINC,4)
#define Contact1  bitRead (PINC,5)

#define Monter    PORTD |= 0x08; PORTD &= ~0x08
#define Descendre PORTD |= 0x04; PORTD &= ~0x04
#define Stop      PORTD &= ~0x0C
```

Clignoter les 3 Leds

L'application considérera que les 3 Leds liées aux contacts d'étage sont des entrées. Il faut donc une initialisation différente pour jouer maintenant avec ces Leds. Ignorons le moteur pour l'instant.

```
//KiAsc1.ino  Clignote les leds
#include "AscDef.h"
void setup ()  {
  pinMode (PinLed1,OUTPUT) ;
  pinMode (PinLed2,OUTPUT) ;
  pinMode (PinLed3,OUTPUT) ;
}
```

```
int att = 500;
void loop () {
  Led3On; delay (att);
  Led2On; delay (att);
  Led1On; delay (att);
  Led3Off; delay (att);
  Led2Off; delay (att);
  Led1Off; delay (att);
}
```

Pour tester les contact, le plus simple est de les copier sur les Leds

```
//KiAsc2.ino
#include "AscDef.h"

void setup () {
// par défaut, non declares en
entree ( Appels, Contacts)
  pinMode (PinLed3,OUTPUT) ;
  pinMode (PinLed2,OUTPUT) ;
  pinMode (PinLed1,OUTPUT) ;
  pinMode (PinMotUp,OUTPUT) ;
  pinMode (PinMotDown,OUTPUT) ;
}
```

Le plus souvent, tester la valeur d'une ligne ou d'une variable se fait avec un `if`.
L'écriture avec un digitalWrite oblige à se souvenir que les Leds sont allumées avec un zéro, (ajouter le !)

```
void loop ()
{
  if (Appel3) Led3On ;  else Led3Off ;
  if (Appel2) Led2On ;  else Led2Off ;
  if (Appel1) Led1On ;  else Led1Off ;
}
```

On peut aussi copier directement- essayez !

```
 digitalWrite (PinLed3,!Appel3) ;
 digitalWrite (PinLed2,!Appel2) ;
 digitalWrite (PinLed1,!Appel1) ;
```

Le moteur est commandé par deux bits. Selon les combinaisons des 2 bits, on a 4 comportements pour le moteur.

Monter=0; Descendre=0; 0 0	bloqué	Le programme `Kia3TestMoteur.ino` fait des aller-retour avec arrêt bloqué. Si on presse sur le poussoir Appel1, le moteur s'arrête en roue libre. Qu'est-ce qui sera le mieux pour l'ascenseur ?
Monter=0; Descendre=1; 0 1	tourne dans un sens	
Monter=1; Descendre=0; 1 0	tourne dans l'autre sens	
Monter=1; Descendre=1; 1 1	roue libre	

Parfois, selon le schéma électronique, il faut un 0 et pas un 1 pour activer un moteur. Pour être compatible, on a défini les mots MototOn (=1 ici) et MotorOff.

Le programme `KiPousToMove.ino` permet de faire monter et descendre l'ascenseur en pressant sur les poussoirs Appel2 (monte) et Appel1 (descend).

Ce programme est utile pour vérifier que les contacts d'étage se font au passage de l'ascenseur. Même une impulsion brève sera vue par le processeur, mais il faut qu'elle existe chaque fois !

Ajuster si nécessaire la distance pour que cela fonctionne au mieux, sans bloquer la cabine.

Le programme pour que la cabine monte et descende sans cesse est facile à écrire. On va naturellement compléter ce programme pour que cela s'arrête quand on presse sur le bouton du bas par exemples

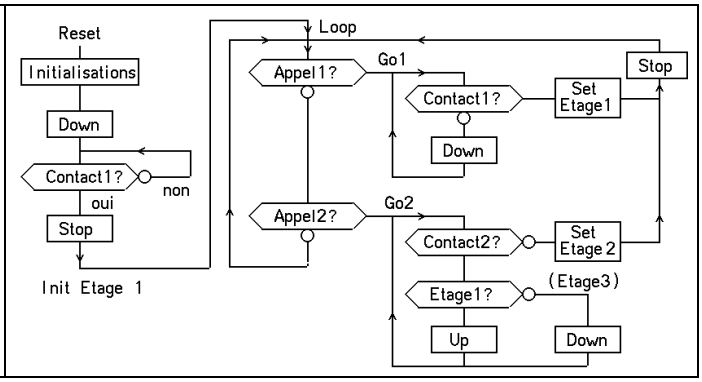
```
// KiAsc3.ino monte-descend selon poussoir
#include "AscDef.h"

void setup () {
  pinMode (PinMotUp,OUTPUT) ;
  pinMode (PinMotDown,OUTPUT) ;
}

void loop () {
  while  (!Contact3) { Monter ;}
  while  (!Contact1) { Descendre ;}
  if     (Appel1) { // lu quand on est en bas
     Stop ;
     while (1) { }  // on reste en bas
  }
}
```

```
// KiAsc4.ino monte-descend
#include "AscDef.h"

void setup ()  {
  pinMode (PinMotUp,OUTPUT) ;
  pinMode (PinMotDown,OUTPUT) ;
}

void loop () {
  while  (!Contact3) {
     Monter ;
     if (Appel1)    { // testé sans cesse
        Stop ;
        while (1) { }
      }
   }
   while  (!Contact1) {
      Descendre ;
      if (Appel1) { // testé sans cesse
        Stop ;
        while (1) { }
      }
   }
}
```

Ascenseur à 2 étages

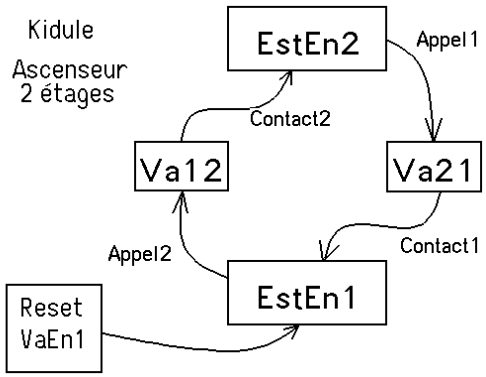
On veut que l'ascenseur monte si on presse sur Appel2, et descende si on presse sur Appel1, et qu'il s'arrête lorsqu'il touche le contact d'étages.
Mais que se passe-il s'il ne s'arrête pas pile sur le contact. Il est monté trop haut au 2e et on repèse sur Appel2. Il monte, et ne s'arrêtera pas !
On voit que le programme doit se souvenir de ce qu'il a fait.

L'approche naïve est d'analyser ce que doit faire le programme pas par pas, et dessiner l'organigramme qui sera traduit en instructions quand on aura bien vérifié le séquencement.
Il faut mémoriser à quel étage on se trouve dans des variables etage1 et 2.
A l'enclenchement, il faut se positionner, par exemple à l'étage 1.



On devine que si l'ascenseur a 3 étages ou plus, l'organigramme devient inextricable. Raisonnons en terme d'état et de transition.

L'ascenseur à 2 étages a 4 états : il est arrêté en haut ou en bas, il monte ou il descend.
S'il est à l'étage 1 et qu'il y a Appel1, on ne fait rien. Si c'est Appel2, on monte.
Maintenant, si on est dans l'état ''monte'' on ignore les boutons d'Appel et on ne regarde que le Contact2. On était en bas et on monte, donc ce contact doit s'activer prochainement et on coupera le moteur pour s'arrêter à l'étage 2. Dans chaque état, un `if` teste la condition qui fera passer à l'état suivant.
Décomposer l'application en états, et coder ce qui fait passer d'un état à l'autre est très puissant. L'instructions `switch .. case` facilite l'écriture. On donne un nom aux états (cas) et on dit ce qu'il faut faire et quel est l'état suivant. Le break; est de rigueur pour ne pas passer par les cas suivants.



Initialisation

A l'enclenchement, on ne sait pas quelle est la position de l'ascenseur, donc dans quel état il se trouve. Il peut aussi se trouver dans un état non prévu, comme par exemple tout en bas, en dessous

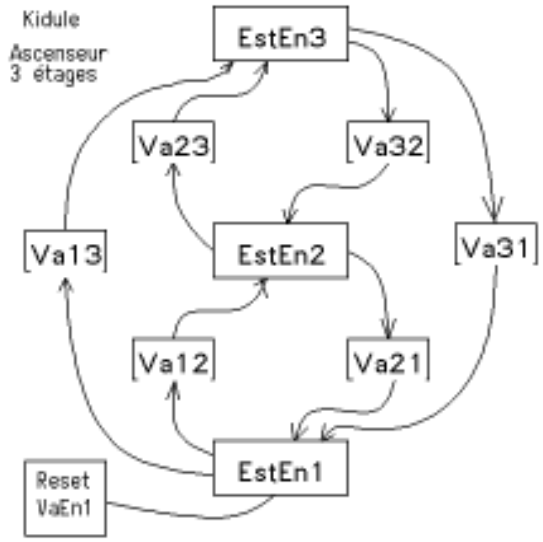
du contact de l'étage 1. Si on demande d'aller à l'étage 1, notre programme le fait descendre, et il ne trouvera jamais le contact1.
L'initialisation choisie est de faire un peu monter l'ascenseur (pas trop s'il est déjà tout en haut) et de le faire ensuite descendre jusqu'à l'étage 1, et d'initialiser la valeur next à ''EstEn1''.

```
// KiAsc2Etages.ino  On monte et descend avec
Appel1 et Appel2
// Utilise switch .. case On a une "machine à 4
états"
#include "AscDef.h"

void setup ()
{
  pinMode (PinMotUp,OUTPUT) ;
  pinMode (PinMotDown,OUTPUT) ;
}
// liste des états
enum {
  EstEn1, EstEn2, Va12, Va21 } next ;
void loop ()
{
  // on se positionne en bas
  Monter  ; // on monte un peu si en-dessous du
contact
  delay (500); // 500 milliseconde --> 2mm
  Stop ;
  while (!Contact1) {
  }
  Descendre ;
  next = EstEn1 ;
```

```
while (1) // boucle principale
  {
    switch(next)  {
     case EstEn1:
      if (Appel2)  {
         Monter ;
         next = Va12;
      }
      break;
    case Va12:
      if (Contact2) {
         Stop ;
         next = EstEn2;
      }
    case EstEn2:
      if (Appel1) {
         Descendre;
         next = Va21;
      }
      break;
    case Va21:
      if (Contact1)  {
         Stop ;
         next = EstEn1;
      } // end if
    }  // end switch
  }  // end while
}  //  end loop
```

Le positionnement initial de l'ascenseur devrait se faire dans le set-up, c'est son rôle. Cela éviterait la boucle while.

Pour l'ascenseur à 3 étages, il suffit d'ajouter des états. Le programme est `KiAsc3Etages.ino`




```
    case EstEn1:
      if (Appel2)  {
        Monter;
        next = Va12;
      }
      else if (Appel3) {
        Monter;
        next = Va13;
      }
```

On pourrait être tenté d'utiliser un OU logique puisque dans les deux cas on doit monter. On écrirait

```
    case EstEn1:
      if (Appel2 || Appel3)  {
        Monter;
```

mais il faudrait ensuite avec des `if` décider de l'état suivant.

Autres applications avec la carte Kidule-Ascenseur
Utiliser un aimant pour activer les capteurs d'étage. On peut alors simuler avec 2 ou trois capteurs une personne qui entre dans une pièce (l'aimant est déplacé vers le haut) ou qui en sort (déplacement vers le bas). On affiche le nombre de personnes dans la pièce.