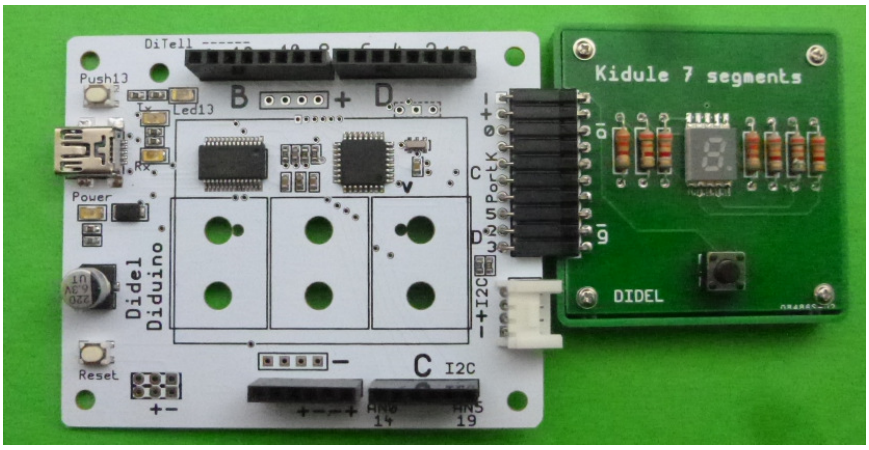
Kidule Affichage 7 segments
Le circuit rassemble les éléments suivants:
1 connecteur mâle MC07-10
1 poussoir
1 affichage 7 segments
7 résistances de 3.3 kOhm (3k3)
1 boitier plastique et 4 vis

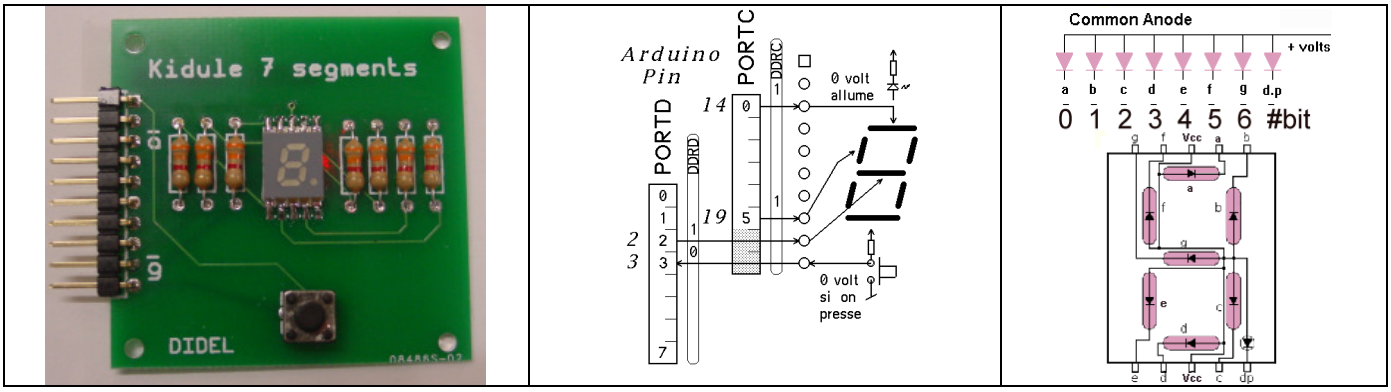


Le Kidule 7-segments se connecte à une carte Diduino, carte compatible Arduino avec un connecteur 8 bits facile à insérer. Ce connecteur est formé avec les pins Arduino A0-A5 et 2,3.

Tous les microcontrôleurs simples travaillent avec des registres 8 bits (ports) sur lesquels on écrit et lit des bytes. Des instructions permettent d'agir sur un bit d'un port (les bits sont numérotés de 0 à 7).
Un port peut être incomplet. Les bits d'un port peuvent avoir plusieurs fonctions. Il faut déclarer comment c'est câblé, et initialiser (configurer) avant de lancer l'exécution.
Arduino a simplifié cette vision en numérotant les pins de la carte, et en cachant la correspondance avec les bits sur les registres.

Kidule Af7seg

Le Kidule seul est facile à comprendre, c'est purement électrique. Il y a 2 fils d'alimentation sur les broches 1 et 2 du connecteur. La tension est de 3 à 5V.
Il y a 7 LEDs avec leur résistance qui limite le courant. Elles sont câblées sur les broches 3 à 9. Un état 0 (proche de 0V) les allume.
Il y a un poussoir, câblé sur la broche 10, qui court-circuite l'entrée vers la masse. La tension est donc 0V si on presse. Une résistance remonte la tension à 5V quand on relâche.



Microcontrôleur

C'est plus compliqué de comprendre la liaison avec la carte Diduino et les ports à l'intérieur du microcontrôleur L'information du Kidule passe par le connecteur, puis par les pins de l'AVR328, et va à l'intérieur du microcontrôleur via des aiguillages.
Le poussoir va sur le PORTD bit 3; il faut définir que c'est une entrée en mettant un zero dans le bit correspondant du registre DDRD (data direction register D). Le segment du milieu va sur le bit2; il faut mettre un 1 sur le bit du registre DDRD pour dire que c'est une sortie.
Arduino ignore les ports. Les pins sont numérotées et trois fonctions interagissent avec les pins.
Dans notre cas le segment "tiret" est sur la pin2 et le poussoir est sur la pin3:

```
pinMode (2,OUTPUT); // met la pin 2 en sortie
pinMode (3, INPUT);   // met la pin 3 en entrée
digitalWrite (2,HIGH); // met la sortie à 5V, donc éteint le segment
```

digitalWrite (2,LOW); // met la sortie à 0V, donc allume le segment
digitalRead (3) // valeur 0 ou 1 à utiliser dans une instruction de test

rappel -- pin (Arduino) - broche (de connecteur) – bit (dans un registre)

Définir

Arduino n'a pas compris la nécessité de bien séparer ce qui est physique et logique.
On vous dit "pour allumer le segment vous écrivez digitalWrite (2,LOW); partout"
Comme il s'agit du segment "tiret", c'est plus clair d'écrire dans le programme "TiretOn" et d'ajouter au début une définition:
#define TiretOn digitalWrite (2,LOW)
Cette ligne de C complète un dictionnaire: TiretOn sera partout remplacé à la compilation par l'instruction Arduino.
Notre premier exemple va copier le poussoir sur le tiret. Il faut définir

```
#define  pinTiret 2
#define  pinPous 3
#define  TiretOn   digitalWrite (pinTiret,LOW)
#define  TiretOff  digitalWrite (pinTiret,HIGH)
#define  PousOn   !digitalRead (pinPous)  // ! = inversion
#define  PousOff   digitalRead (pinPous)
```

Initialiser

Dans l'environnement Arduino, les programmes commencent par une fonction `setup` qui configure le processeur et les périphériques. En particulier, on doit dire si les pins sont en entrée ou sortie.
Notre premier exemple va copier le poussoir sur le tiret. Il faut initialiser ces 2 signaux:

```
void setup() {
   pinMode (pinTiret,OUTPUT);   // met la pin 2 en sortie
   pinMode (PinPous, INPUT);    // met la pin 3 en entrée
}
```

Exécuter

Le programme est écrit dans une fonction loop:

```
void loop() {
   if (PousOn) {TiretOn;}
   else {TiretOff;}
}
```

(programme Ki7seg1.ino)

On peut inverser la condition (éteindre si on presse), ou ajouter un délai. Que se passe-t-il si on écrit

```
void loop() {
   if (PousOn) {delay (1000); TiretOn; delay (1000);}
   else {TiretOff;}
}
```

(programme Ki7seg2.ino)

et

```
void loop() {   //Presser longtemps! - pourquoi?
   if (PousOn) { TiretOn; }
   else { delay (1000); TiretOff;}
}
```

(programme Ki7seg3.ino)

Pour copier le poussoir sur une LED, on peut aussi utiliser un while

```
void loop() {
   while (PousOn)  { TiretOn; }
   TiretOff;
}
```

(programme Ki7seg4.ino)

Répéter dans une boucle la même action est courant en programmation. Détecter que l'on vient de presser et exécuter une seule fois l'instruction TiretOn; est compliqué et inutile (pas d'usure, on ne verrait pas la différence).

Commentaires sur l'Instruction `while (condition)`

Le while (condition) permet de boucler on non selon une condition vrai/faux, donc une valeur différente de zéro ou égale à zéro .

Le groupe d'instructions entre { } est exécuté jusqu'à ce que la condition devienne fausse. Donc `while (1){}` boucle indéfiniment !
Avec le if, on fait si la condition est vraie et on continue (instruction non bloquante). Avec le while, on se bloque à répéter tant que la condition est vraie. Le signe de négation ! permet de changer vrai en faux.

Action alternée

Une première action sur le poussoir doit allumer le tiret, une seconde action éteindre.

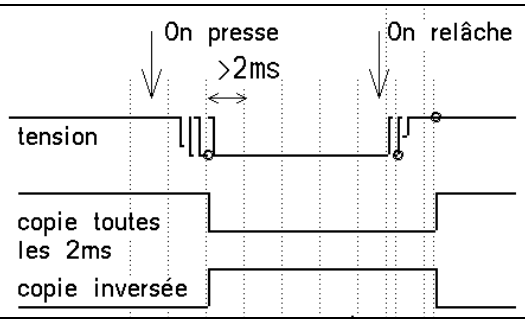
```
void loop() {
   while (PousOff) {} // on attend l'action
   TiretOn;           // on allume
   while (PousOn) {}  // on attend le relâchement
   // cela reste allumé
   while (PousOff) {} // on attend l'action
   TiretOff;          // cette fois on éteint
   while (PousOn) {}  // on attend le relâchement
}
```

programme Ki7seg5.ino)

Le comportement n'est pas fiable parce qu'il y a des rebonds de contact. Il faut ajouter des délais de 2 à 20 millisecondes dans les boucles while.

```
while (PousOff) [delay (20);}
```

programme Ki7seg6.ino)



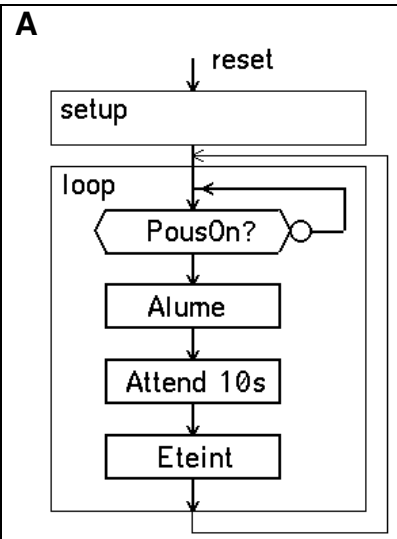
Minuterie d'escalier

A

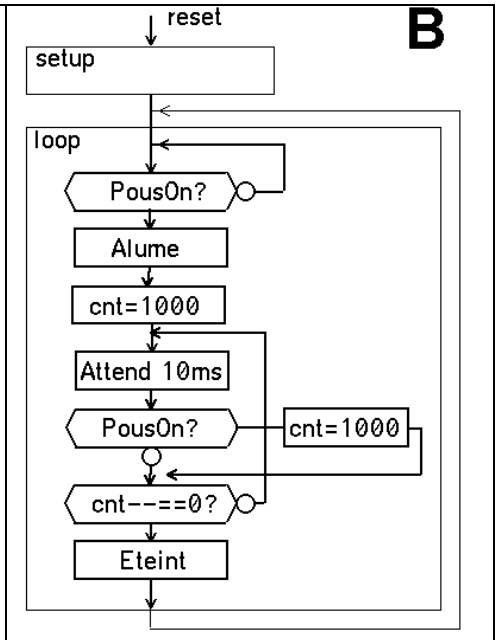
On programme pour que le segment reste allumé 10 secondes. Que se passe-t-il si on represse pendant les 10 secondes?
Modifions le programme pour que l'on redonne 10 secondes à chaque pression.

B

Ce n'est pas évident car le délai est bloquant. Il faut tester régulièrement le poussoir et remplacer l'attente de 10 secondes par 1000 attentes de 10ms, avec test chaque fois, et réinitialisation du délai si le poussoir est actif.



programme Ki7seg5.ino



Le programme B qui réactive l'attente, sans les définitions et set-up est:

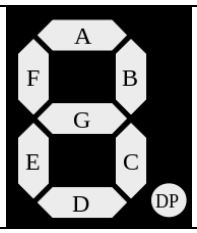
```
void loop() {
   while (PousOff) {delay (20);} // on attend l'action
   TiretOn;           // on allume
   cnt = 1000;
   while (cnt--) {  // teste et décompte
     delay (10);
     if(PousOn) [cnt = 1000;}
   }
   TiretOff;          // cette fois on éteint
}
```

programme Ki7seg5.ino)

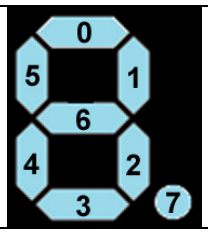
Affichons des chiffres

Le segment a est câblé sur le bit 0, son poids est 1

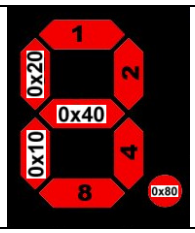
Le segment g est câblé sur le bit 6, son poids est 40 en hexa, 64 en décimal



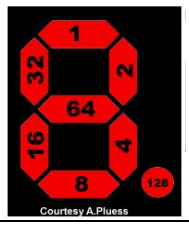
noms



no de bits



poids en hexa



poids en décimal

Pour former un 4 on allume B C F G donc les bits 1 2 5 6 sont actifs (à 0V pour notre circuit). Le code est 01100110=0x66.
Le microprocesseur devra envoyer l'inverse (10011001).
On peut raisonner avec les poids en décimal: 2+4+64+32 = 102

Table des segments

s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

```
#define Zero  0x3F
#define Un    0x06
#define Deux  0x5B
#define Trois 0x4F
#define Quatre 0x66
#define Cinq  0x6D
#define Six   0x7D
#define Sept  0x07
#define Huit  0x7F
#define Neuf  0x6F
```

Définition de la table

Taseg [10] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F };
Difficulté: Le PORTC a 6 bits, le 7e bit est sur la pin 2
Le truc est le suivant: On indexe la table, le résultat est mis dans la variable `seg`. On copie cette variable dans le PORTC. 6 bits sont ok. On teste le 7e et on le copie sur `pinTiret`

```
seg = TaSeg(chiffre);
PORTC = ~seg    // les segnents s'allument pour 0Volt
if (seg&0b01000000) {TiretOn;}
else {TiretOff;}
```

Compter de 0 à 9

il faut compléter le setup, déclarer la table et les variables.

```
// Ki7seg9.ino On compte de 0 à 9
#define  PinTiret 2
#define  PinPous 3
#define  TiretOn   digitalWrite (2,LOW)
#define  TiretOff   digitalWrite (2,HIGH)
#define  PousOn  !digitalRead (3)
#define  PousOff  digitalRead (3)

void setup() {
  pinMode (PinTiret, OUTPUT);  // met la pin 2 en sortie
  pinMode (PinPous, INPUT);    // met la pin 3 en entrée
  DDRC = 0xFF;
}
byte chiffre = 4;
byte seg;
byte TaSeg [] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F };
void loop() {
  chiffre = 0;
  while (chiffre < 10) {
    seg = TaSeg[chiffre];
    PORTC = ~seg ;   // les segments s'allument pour 0 Volt
    if (seg & 0b01000000) { TiretOn; }
    else { TiretOff; }
    chiffre++;
    delay (1000);
  }
}
```

Qu'avons-nous compris?
Que les programmes ont une partie configuration (setup) et une partie exécution (algorithme et interaction avec l'utilisateur). Cette 2e partie ne doit pas dépendre du câblage,
Pour bien comprendre un langage, une application, il faut procéder par petits pas, tester des variantes, chercher à optimiser.

jdn 160310/180602