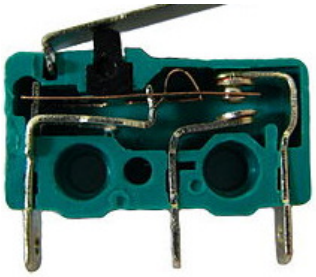
http://www.didel.com/ info@didel.com

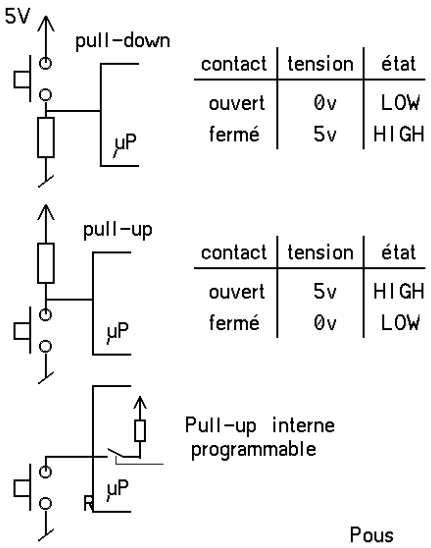
www.didel.com/diduino/Composants02.pdf

Composants électroniques avec le Diduino
Comment interfacer, comment programmer

C10 Poussoirs et commutateurs

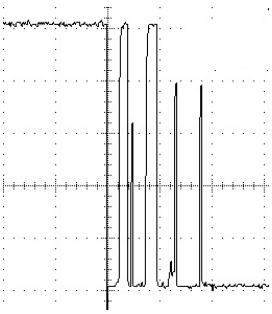


Par poussoir on entend aussi contact de fin de course, interrupteurs, toute solutions mécaniques qui laissent passer le courant ou non. Pour lire l'état d'un contact simple, il faut câbler un diviseur de tension avec une résistance, pull-up ou pull-down. Ne pas utiliser le câblage en pull-down: les processeurs ont sur la plupart de leurs pins des pull-up programmable, de 20 à 50 kOhm. Mais on ne sait pas quel est le mode initialisé, et activer ou désactiver les pull-up interne suppose des bonnes connaissances du processeur. Lire l'état d'un poussoir et copier cet état sur une Led est facile. Voir Cours02.pdf pour des exemples. Il faut maintenant bien comprendre la dynamique entre les deux états.



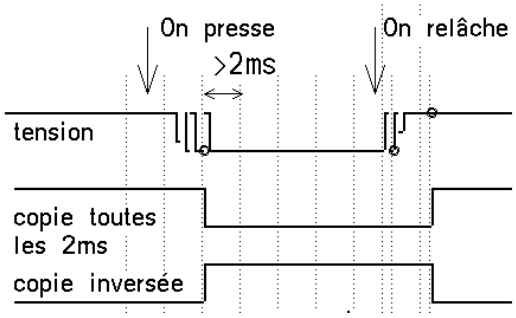
Le problème de tous les contacts mécanique est que le contact n'est pas brusque; il y a parfois rebondissement de la pièce mobile au moment du choc, ce qui veut dire que l'état oscille pendant quelques millisecondes avant de se stabiliser. Il faut un oscilloscope pour voir ces rebonds. On sait les supprimer avec un filtre électronique, mais si on peut le faire par programmation, c'est moins encombrant et moins coûteux.
Si on a câblé une Led allumée par un LOW sur la pin 12 par exemple, et un poussoir avec pull-up sur la pin 12, quand on écrit

```
digitalWrite ( Led, digtalRead (Poussoir));
```



la Led clignote selon les rebonds, mais notre oeil ne le voit pas. Mais un programme compteur d'actions sur le poussoir va les voir et les compter! (exemple avec un kidule dans xxx).

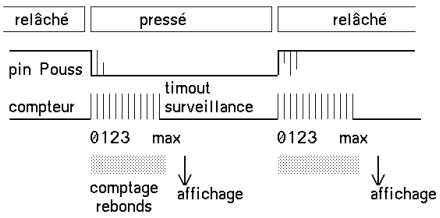
Pour passer par dessus les rebonds, il suffit de lire le signal avec une période supérieure aux rebonds, 1 à 5 millisecondes pour un contact bien construit.
Donc entre chaque lecture, un delay (2ms); est adéquat.




```
void loop ()
{
  delay (2);
  if  (digtalRead (Poussoir)==LOW)
  {
  digitalWrite ( Led, LOW) ;
  }
}
```

Pour compter les actions et se débrancher dans des programmes de démonstration, voir le document xx (en préparation adapté de www.didel.com/kidules/Poussoir.pdf) . Ce document propose des fonctions qui facilite la gestion d'un poussoir.

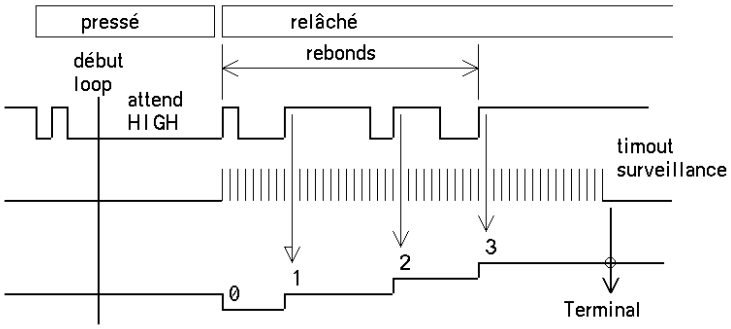
La question intéressante maintenant, c'est comment compter ces rebonds? Procédons par étapes.



On veut lancer un programme qui compte les rebonds et affiche le nombre de rebonds sur le terminal. Le programme attend que l'on presse, surveille le poussoir et affiche. Comment lui dire d'attendre de surveiller. On ne peut pas utiliser delay() qui bloque tout. Il faut un compteur dans la boucle de surveillance, et on sort quand ce compteur dépasse une valeur limite. Cela s'apelle programmer un time-out (foreclos en français).
Le cœur du programme a donc l'allure suivante pour surveiller quand on relâche (les rebonds sont plus abondants) :

```
int cntTimout =0;  // valeur max 30'000
MaxTimout = 30000;
…  // définition Poussoir, setup
void loop ()
{
  while (digitalRead(Poussoir) == LOW) { } //on attend un HIGH
  cntTimout = 0 ;
  while (cntTimout < MaxTimout)
  {
     delayMicroseconds (10) ;
     cntTimout++ ;
     // xx - on surveillera le signal et comptera les rebonds
  }
  // yy - on affichera le compteur de rebonds
  while (digitalRead(Poussoir) == HIGH) { } //on attend un LOW pour recommencer
}  // fin du loop
```

Il faut tester ce programme avant d'aller plus loin, en mettant à la place de xx et yy allumer une Led et éteindre une Led. On verra la Led s'allumer 0.3s à chaque relâchement (programme TestRebond0.ino)

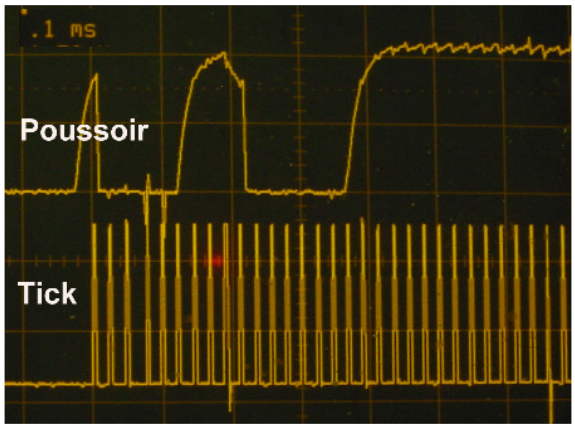


Comment surveiller? Si le poussoir est HIGH, on a relâché, on attend le timout. S'il passe à LOW on compte, mais le problème est que l'on veut compter les rebonds, et pas le temps passé dans les rebonds. Il faut activer un flag pour dire, on a compté le passage à un, il ne faut pas compter les suivants.

```
if (Poussoir == LOW)
{
   flag = LOW)
}
if (Poussoir == HIGH)
{
   if (flag == LOW)
   {
      cntRebonds++ ;
      flag = HIGH;
   }
}
```

On voit le petit jeu de ce flag, remis à zéro quand on attend un rebond, et qui s'active quand on a compté le rebond. Il n'y a qu'à remplacer la partie xx.
L'affichage sur terminal (partie xx est évident : `Serial.println (cntRebonds);`

Deux instructions ont été ajoutées dans le programme final, pour observer à l'oscilloscope la période d'échantillonage. Le signal Tick est activé/désactivé dans deux digitalWrite consécutifs (largeur 4 microsecondes) dès que le programme sort du while qui attend le relâchement. Comme le montre la figure ci-contre, prise pour le meilleur rebond observée (le plus souvent il n'y en a pas), la période d'échantillonage est de 25 microsecondes (une centaine d'instructions du processeur) et la durée des rebonds dans ce cas est de 0.5 ms.

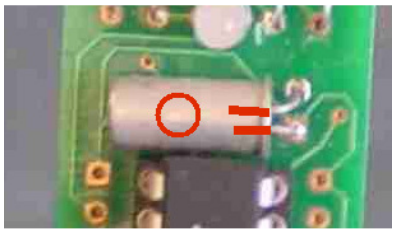


Programme final sous TestRebond.ino

Ce programme est un bon exemple de programmation temps réel, où on se préoccupe du temps mis pour exécuter une partie de programme. A part cela, pour vous rassurer ou vous inquiéter, ce n'est pas un programme difficile, mais l'interaction d'un programme avec des phénomènes en temps réél est toujours difficile à bien maîtriser.

C11 Inclinomètre et capteurs d'orientation

L'inclinomètre le plus simple (tilt sensor) est formé d'une bille qui se déplace dans un tube et vient toucher des contacts. Ce capteur se gère comme un poussoir, mais les rebonds de contact peuvent être importants, et il faut les distinguer des rebonds de la bille. Ils ne mesurent pas une inclinaison précise, et sont utilisés le plus souvent comme détecteur de chocs



Les accéléromètres que l'on trouve dans les tablettes donnent trois valeurs à partir desquelles on peut calculer une inclinaison par rapport à la verticale. L'interface est SPI ou I2C, décrits dans Composants05.pdf. Des librairies existent et si la mise en œuvre de l'interface électronique est facile, le logiciel est délicat et il vaut mieux acheter des modules.

Suite: www.didel.com/diduino/Composants03.pdf