CAPTEURS / COMPOSANTS / ELECTRONIQUE

Le bouton poussoir

PAR CFAURY · PUBLIÉ 30 NOVEMBRE 2017 · MIS À JOUR 4 JUILLET 2019

Un **bouton poussoir** est un interrupteur (ou contacteur) **monostable** : il n'est stable que dans la position « relâché ».

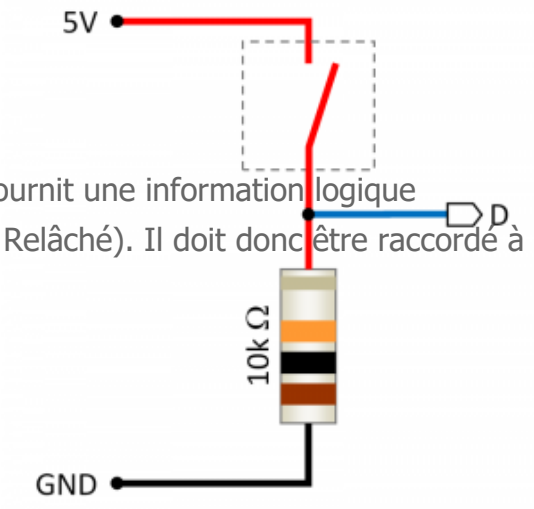
Il peut être :

- à ouverture = fermé au repos :

- à fermeture = ouvert au repos :

Câblage

Un
bouton



poussoir fournit une information logique (Appuyé / Relâché). Il doit donc être raccordé à un port Numérique en mode INPUT.

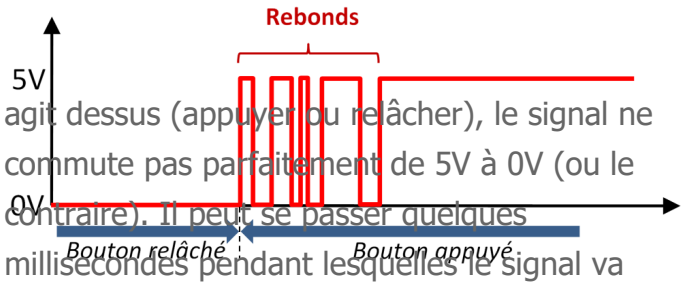
Le montage ci-contre permet de fournir au port numérique D :

- 0V quand le bouton est relâché : état bas = `LOW`
- 5V quand le bouton est appuyé : état haut = `HIGH`

> Remarque : quand le bouton est relâché, Pour ne pas laisser flottante l'entrée du port (cas d'un interrupteur mécanique par exemple), il est nécessaire d'utiliser une résistance de rappel (pull down) ou de tirage (pull up) .

Programmation

Un bouton n'est pas un objet parfait : lorsqu'on

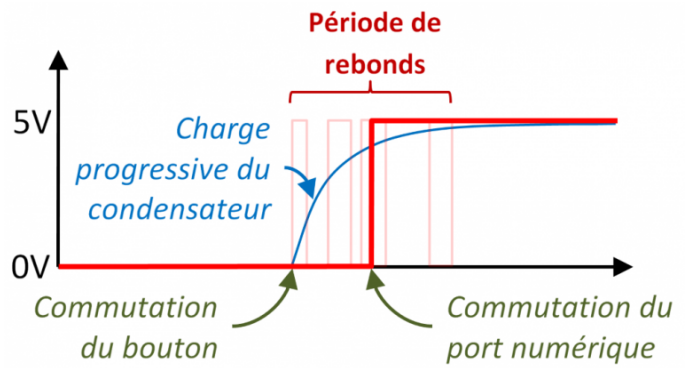


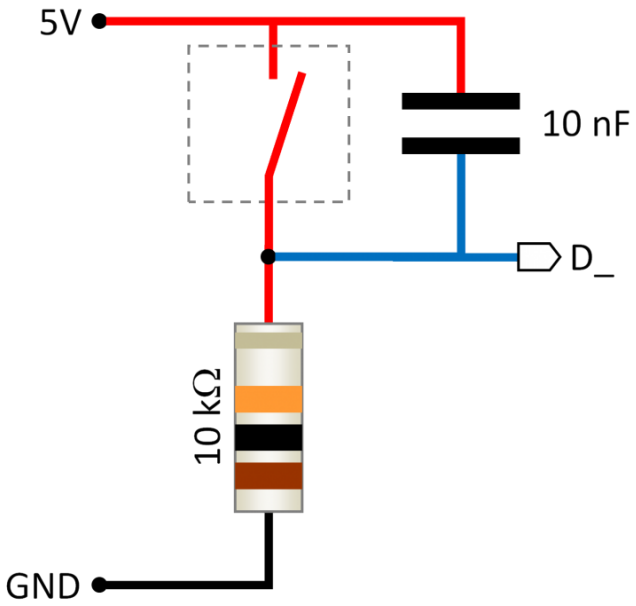
agit dessus (appuyer ou relâcher), le signal ne commute pas parfaitement de 5V à 0V (ou le contraire). Il peut se passer quelques millisecondes pendant lesquelles le signal va « rebondir » avant de se stabiliser. Ce rebond peut provoquer un comportement inattendu du programme (comptage d'appuis en trop, arrêt ou démarrage prématuré de procédures, …).

Il existe diverses solutions pour éviter cela…

En filtrant le signal

Une des solutions est d'utiliser un condensateur en parallèle avec le bouton. Ce composant permet d'absorber les rebonds en se chargeant (ou se déchargeant) progressivement. La commutation du port numérique n'a lieu qu'une fois le condensateur suffisamment chargé, soit un peu après la commutation du bouton.





Avec ce montage on peut programmer l'utilisation d'un bouton de la manière suivante :

```
int pin_LED = 13;   // port numérique associé
int pin_button = 2; // port numérique lié au b
int state = LOW; // variable d'état de la LED

void setup() {
   pinMode(pin_LED, OUTPUT);   // réglage du p
   pinMode(pin_button, INPUT); // réglage du p
}

void loop() {
   if (digitalRead(pin_button) == HIGH) { // s
      state = !state; // ... inversion de la v
   }
   digitalWrite(pin, state); // action sur la

   // Suite du programme ... simulé par une in
   delay(100);
}
```

Il reste néanmoins un inconvénient à cette solution : si la durée d'exécution de la boucle `loop()` est trop long, il peut arriver qu'un utilisateur appuie pendant un temps si court que l'instruction qui test de l'état du bouton (`if (digitalRead(pin_button) == HIGH)`) n'a pas le temps de « voir » pas cet appui.

> Activité : tester le programme ci-dessus avec une attente de 1s … et constater le

problème ...

En utilisant une interruption

Les microcontrôleurs Arduino possèdent des ports supportant les **interruptions matérielles** : une commutation d'un de ces ports (programmé pour cela), provoque l'arrêt de l'exécution du programme pour faire une tâche particulière, puis retourner à l'exécution normale du programme.

```
int pin_LED = 13;     // port numérique associé
int pin_button = 2;   // port numérique lié au b
volatile int state = LOW; // variable d'état de

void setup() {
   pinMode(pin_LED, OUTPUT);     // réglage du p

   // Création de l'interruption
   attachInterrupt(0, blink, CHANGE);
}

void loop() {
   digitalWrite(pin_LED, state); // action sur

   // Suite du programme ... simulé par une ins
   delay(100);
}

void blink() {
   state = !state; // inversion de la variable
}
```

Dans l'exemple précédent, un changement d'état du bouton (constante `CHANGE`), provoquera aussitôt l'arrêt de l'exécution du programme pour exécuter la fonction `blink()` , et ensuite retourner au programme…

👉 VOUS AIMEREZ AUSSI...