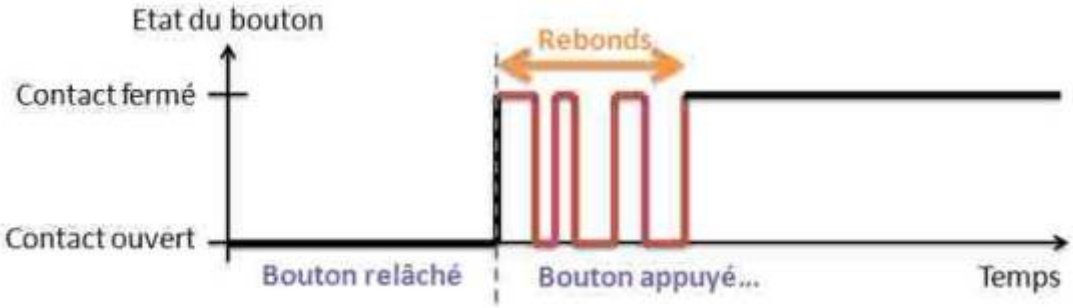
: Les Boutons poussoirs : l'anti rebond

Un bouton poussoir ou un interrupteur, va pendant une durée généralement inférieure à la milliseconde, osciller plusieurs fois entre deux états :

Illustration signal rebonds



Créeation de deux variables qui vont garder en mémoire l'état présent et passé du capteur.

Lire l'entrée digitale et valider son état en fonction de l'état précédent et d'un délai anti-rebond.

La méthode peut être implémentée avec la fonction delay() mais l'utilisation de la **fonction millis() est plus propre et plus adéquate.**

anti_rebond.ino

```
const int didPin  = 2;
//Variables
bool didStatus  = false;
bool oldDidStatus  = false;
unsigned long lastDebounceTime  = 0;
unsigned long debounceDelay  = 50;
void setup() {
 //Init Serial USB
 Serial.begin(9600);
 Serial.println(F("initialisation"));
 //Init digital input
 pinMode(didPin, INPUT_PULLUP);
}
void loop() {
 debounceDid();
}
void debounceDid( ) { /* function debounceDid */
 ////debounce DigitalDebounce
 int reading = digitalRead(didPin);
 if (reading != oldDidStatus) {
   lastDebounceTime = millis();
 }
```

```
 if ((millis() - lastDebounceTime)>= debounceDelay) {
   if (reading != didStatus) {
     didStatus = reading;
     Serial.print(F("Etat du Bouton : ")); Serial.println(didStatus);
   }
 }
 oldDidStatus = reading;
}
```

automaintient.ino

```
boolean etatLed; // Déclaration de la variable etatLed

void setup(){
    pinMode(1, INPUT); // L'entrée du bouton poussoir
    pinMode(2, OUTPUT); // La sortie de la LED
    etatLed = false; // Initialisation de la variable etatLed
}

void loop(){
    if(digitalRead(1) == HIGH){ // Si un appui sur le poussoir est
détecté
        etatLed = !etatLed; // On inverse l'état de la variable etatLed
    }
    digitalWrite(2, etatLed); // On active ou pas la sortie en fonction
de etatLed

}
```

automaintient_antirebond

```
/*
- Ce code sert d'auto-maintien et d'anti-rebond pour bouton poussoir
monté en pullup.
- Les variables sont à déclarer en début de programme. Adapter leur nom
au programme si besoin.
- La fonction setup doit être ignorée, elle ne sert que pour le
fonctionnement de ce programme de manière autonome.
- La fonction loop représente le programme de destination et ne sert
qu'à appeler la fonction "antiRebondAvecVerrouillage" et à représenter
son résultat sur une sortie. Elle n'est pas à copier.
- La fonction "antiRebondAvecVerrouillage":
    - Est à copier dans le programme et à renommer si besoin.
    - Prend comme argument le numéro de l'entrée à lire ou la variable
correspondante.
    - Retourne une variable de type int qui représente l'état du
contact auto-maintenu.
*/
```

```
const int POUSSOIR = 2; //Constante, ne sert que pour ce programme, ne pas intégrer.
int out = 0;
boolean verrouillage = 0;
unsigned long int t1;

void setup(){
    pinMode(POUSSOIR, INPUT_PULLUP);
    pinMode(10, OUTPUT);
}

void loop(){
    int outFonction = antiRebondAvecVerrouillage(POUSSOIR);
    if (outFonction == 1){
        digitalWrite(10, HIGH);
    }
    if (outFonction == 0){
        digitalWrite(10, LOW);
    }
}

int antiRebondAvecVerrouillage(int inPoussoir){

    if(digitalRead(inPoussoir) == HIGH){
        t1 = 0;
        verrouillage = 0;
    }

    if((digitalRead(inPoussoir) == LOW) and (t1 == 0)){
        t1 = millis();
    }

    if((digitalRead(inPoussoir) == LOW) and (millis() - t1 >= 50) and (verrouillage == 0)){
        switch(out){
            case 0:
                out = 1;
                verrouillage = 1;
                break;
            case 1:
                out = 0;
                verrouillage = 1;
                break;
        }
        return out;
    }
}
```

Utiliser une librairie anti-rebond

OneButton.h

Cette bibliothèque Arduino améliore l'utilisation d'un seul bouton pour l'entrée.

Il montre comment utiliser une broche d'entrée numérique avec un seul bouton-poussoir attaché pour détecter certains des événements de pression de bouton typiques comme les clics simples, les doubles clics et les pressions prolongées.

Cela vous permet de réutiliser le même bouton pour plusieurs fonctions et réduit les investissements matériels.

Clonez ce référentiel dans Arduino/Librariesou utilisez le gestionnaire de bibliothèque IDE Arduino intégré pour installer une copie de cette bibliothèque. Vous pouvez trouver plus de détails sur l'installation des bibliothèques ici, sur le site Web d'Arduino .

1.ino

```
# inclure  < Arduino.h >
# inclure  < OneButton.h >
```

Chaque bouton physique nécessite sa propre OneButtoninstance. Vous pouvez les initialiser comme ceci :

2.ino

```
#define BUTTON_PIN 4

/**
 * Initialize a new OneButton instance for a button
 * connected to digital pin 4 and GND, which is active low
 * and uses the internal pull-up resistor.
 */

OneButton btn = OneButton(
  BUTTON_PIN,  // Input pin for the button
  true,        // Button is active LOW
  true         // Enable internal pull-up resistor
);
```

3.ino

```
#define BUTTON_PIN 4

/**
 * Initialize a new OneButton instance for a button
 * connected to digital pin 4, which is active high.
 * As this does not use any internal resistor
```

```
 * an external resistor (4.7k) may be required to create a LOW signal
when the button is not pressed.
 */

OneButton btn = OneButton(
  BUTTON_PIN,  // Input pin for the button
  false,       // Button is active high
  false        // Disable internal pull-up resistor
);
```

Joindre des événements d'état

Une fois votre bouton initialisé, vous pouvez gérer les événements en les attachant à l'instance du bouton. Les événements peuvent être des fonctions statiques ou des lambdas (sans variables capturées)

4.ino

```
// Handler function for a single click:
static void handleClick() {
  Serial.println("Clicked!");
}

// Single Click event attachment
btn.attachClick(handleClick);

// Double Click event attachment with lambda
btn.attachDoubleClick([]() {
  Serial.println("Double Pressed!");
});
```

N'oubliez pas tick()!

Pour que OneButton fonctionne correctement, vous devez appeler chaque instance de boutontick() dans votre loop().

5.ino

```
void loop() {
  btn.tick();

  // Do other things...
}
```

Événements d'État

Voici une liste complète des événements gérés par cette bibliothèque :

Attacher la fonction	La description
attachClick	Se déclenche dès qu'un seul clic est détecté.
attachDoubleClick	Se déclenche dès qu'un double clic est détecté.
attachMultiClick	Se déclenche dès que plusieurs clics ont été détectés.
attachLongPressStart	Se déclenche dès que le bouton est maintenu enfoncé pendant 1 seconde.
attachDuringLongPress	Se déclenche périodiquement tant que le bouton est maintenu enfoncé.
attachLongPressStop	Se déclenche lorsque le bouton est relâché après un long maintien.

Calendrier de l'événement

Les événements valides se produisent lorsque tick()est appelé après un nombre spécifié de millisecondes.

Vous pouvez utiliser les fonctions suivantes pour modifier la synchronisation.

Remarque : Attacher un double-clic augmentera le délai de détection d'un simple clic.

Si un événement de double-clic n'est pas attaché, la bibliothèque assumera un simple clic valide après une durée d'un clic, sinon elle doit attendre que le délai d'expiration du double-clic soit écoulé.

Une fonction	Défaut	La description
setDebounceTicks(int)	50 msec	Période de temps pendant laquelle ignorer les changements de niveau supplémentaires.
setClickTicks(int)	500 msec	Délai d'attente utilisé pour distinguer les clics simples des doubles clics.
setPressTicks(int)	800 msec	Durée de maintien d'un bouton pour déclencher un appui long.

Vous pouvez modifier ces valeurs par défaut, mais sachez que lorsque vous spécifiez des durées trop courtes, il est difficile de cliquer deux fois ou vous créerez une pression au lieu d'un clic.

From:
https://chanterie37.fr/fablab37110/ - **Castel'Lab le Fablab MJC de Château-Renault**

Permanent link:
https://chanterie37.fr/fablab37110/doku.php?id=start:arduino:bp&rev=1649419969

Last update: **2023/01/27 16:08**