Le bouton-poussoir récalcitrant

Dans ce montage, vous verrez qu'un bouton-poussoir, ou aussi un interrupteur, ne se comporte pas toujours comme vous l'auriez voulu. Prenons pour exemple un bouton-poussoir qui, en théorie, ferme le circuit tant qu'il reste enfoncé, et le rouvre quand il est relâché. Rien de neuf en soi et rien de bien difficile à comprendre. Mais les circuits électroniques, dont le rôle consiste par exemple à déterminer le nombre exact de pressions sur le bouton-poussoir pour une exploitation ultérieure, posent un problème dont on ne peut se douter au départ.

Une histoire de rebond

Le mot-clé de ce montage est rebond. Appuyer sur un bouton-poussoir normal et même le maintenir enfoncé revient à fermer le contact mécanique une seule et unique fois dans le bouton-poussoir. Ce n'est pourtant pas le cas la plupart du temps, car le composant en question ouvre et referme plusieurs fois le contact dans un intervalle de temps très court, de l'ordre de la milliseconde. Les surfaces de contact d'un bouton-poussoir ne sont en général pas complètement lisses, et on peut voir de multiples aspérités et impuretés quand on les observe au microscope électronique. Ainsi, les points de contact des matériaux conducteurs ne se touchent pas instantanément et pas durablement au moment du rapprochement. L'effet en question peut aussi être obtenu par vibration ou montage sur ressorts du matériau, le contact étant alors brièvement fermé puis de nouveau ouvert plusieurs fois l'une derrière l'autre lors de la jonction.

Ces impulsions délivrées par le bouton-poussoir sont enregistrées et traitées en bonne et due forme par le microcontrôleur, c'est-à-dire comme si vous appuyiez très vite et très souvent sur le bouton-pous-

soir. Ce comportement est bien sûr gênant et doit être évité d'une manière ou d'une autre. Regardons maintenant le chronogramme de plus près.

Figure 4-1 ▶
Bouton-poussoir à rebond

J'ai appuyé une seule fois sur le bouton-poussoir et l'ai ensuite maintenu appuyé, mais celui-ci a interrompu plusieurs fois la liaison souhaitée avant que l'état stable de la connexion ne soit atteint. Cette suite de fermetures et d'ouvertures du circuit jusqu'à ce que le niveau `HIGH` définitif souhaité soit atteint est appelée rebond. Ce comportement peut aussi se constater dans l'opération inverse. Si je relâche le bouton-poussoir, plusieurs impulsions peuvent éventuellement être générées jusqu'à ce que j'obtienne enfin le niveau `LOW` souhaité. Le rebond du bouton-poussoir est à peine perceptible voire carrément invisible par l'œil humain et si un circuit censé commander une LED quand le bouton-poussoir est enfoncé était construit, les différentes impulsions se verraient comme un niveau `HIGH` du fait de la persistance oculaire. Essayons donc une autre solution. Nous pourrions construire un circuit avec un bouton-poussoir sur une entrée numérique et une LED sur une autre sortie numérique.

Mais il n'y a rien de nouveau là-dedans. Qu'est-ce que ça apporte ? Vous venez de dire que ce type de rebond possible ne se voyait pas dans un circuit.

Notre circuit n'est pas le seul composant. Il y a certes le matériel mais il y a aussi le logiciel et nous entendons le configurer de telle sorte que la LED s'allume à la première impulsion. Elle doit s'éteindre à l'impulsion suivante et se rallumer à celle d'après et ainsi de suite. Nous avons donc affaire à une bascule du niveau logique. Si maintenant plusieurs impulsions sont enregistrées par le circuit ou plutôt par le logiciel quand le bouton-poussoir est appuyé, la LED change alors plusieurs fois d'état.

Dans le cas d'un bouton-poussoir sans rebond, les états doivent être tels que représentés dans le diagramme de la figure 4-2.

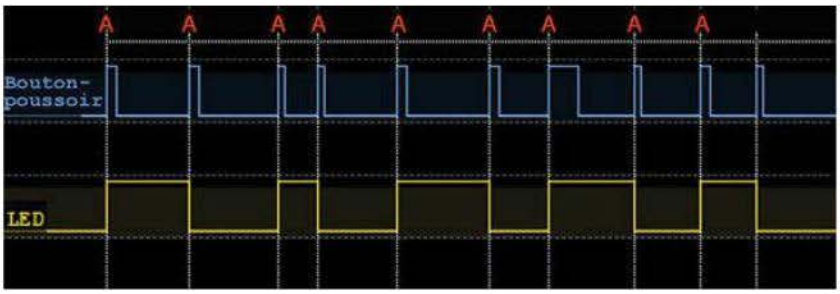


Figure 4-2
Changement du niveau de la LED pour un appui sur le bouton-poussoir

On voit que dans le cas de multiples appuis sur le bouton-poussoir (flanc montant), lesquels sont ici indiqués par un A, l'état de la LED bascule. Comment faire pour que le logiciel se charge de l'exécution ? Voyons maintenant la liste des composants.

Composants nécessaires

Pour le circuit suivant, j'ai trouvé un ancien bouton-poussoir dans mon bric-à-brac qui, lui, rebondira à coup sûr énergiquement. Les nouveaux boutons–poussoir disponibles peuvent présenter une protection mécanique contre le rebond avec un point d'appui reconnaissable. Quand on appuie dessus, on peut entendre un léger claquement. Cela indique que le contact a été fermé avec une pression ou une vitesse plus élevée de manière à empêcher ou minimiser le rebond.

	1 LED rouge
	1 bouton-poussoir (sans protection antirebond)
	1 résistance de 330 Ω
	1 résistance de 10 kΩ
	Plusieurs cavaliers flexibles de couleurs et de longueurs diverses

Code du sketch

Le code du sketch est le suivant pour cet exemple.

```
int buttonPin = 2;   //Bouton-poussoir en broche 2

int buttonValue = 0; //Variable pour enregistrer l'état du bouton-
                     //poussoir
int previousButtonValue = 0; //Variable pour enregistrer l'ancien
                             //état du bouton-poussoir
int ledPin = 8;              //LED en broche 8
int counter = 0;             //Variable de compteur
void setup(){
  pinMode(buttonPin, INPUT); //Broche bouton-poussoir comme entrée
  pinMode(ledPin, OUTPUT);   //Broche LED comme sortie
}

void loop(){
  buttonValue = digitalRead (buttonPin); //Interrogation du
                                         //bouton-poussoir
  //La valeur précédente du bouton-poussoir est-elle différente
  //de la valeur actuelle ?
  if(previousButtonValue != buttonValue){
     if(buttonValue == HIGH){
       counter++;          //Incrémentation du compteur (+1)
     }
   }
   previousButtonValue = buttonValue; //Sauvegarde de la valeur
                                      //actuelle du bouton-poussoir
   if(counter % 2 == 0)     //La variable du compteur est-elle un
                            //nombre pair ?
   digitalWrite(ledPin,HIGH);
else
   digitalWrite(ledPin,LOW);
}
```

Le code ne semble pas très compliqué à première vue, mais il est cette fois-ci un peu plus subtil. Vous allez bientôt voir dans quelle mesure.

Revue de code

On commence ici aussi par déclarer et initialiser une série de variables globales.

Variable	Objet
buttonPin	Cette variable contient le numéro de broche pour le bouton-poussoir (2).
buttonValue	Cette variable enregistre l'état du bouton-poussoir.
previousButtonValue	Cette variable sert à enregistrer l'état précédent du bouton-poussoir.
ledPin	Cette variable contient le numéro de broche pour la LED (8).
Counter	Cette variable mémorise les niveaux HIGH de l'état du bouton-poussoir.

Tableau 4-1
Variables nécessaires et leur objet

L'initialisation des différentes broches au sein de la fonction `setup` ne nécessitant aucune explication supplémentaire, passons directement à la fonction `loop`. Le niveau sur la broche du bouton-poussoir est continuellement interrogé via la fonction `digitalRead` et sauvegardé dans la variable `buttonValue` :

```
buttonValue = digitalRead(buttonPin) ;
```

La tâche du sketch consiste cependant à détecter chaque appui – représenté par un niveau `HIGH` – sur le bouton-poussoir, et à incrémenter en conséquence une variable de compteur. Normalement, les lignes de code suivantes devraient suffire.

```
void loop(){
  buttonValue = digitalRead(buttonPin); //Interrogation du bouton-
                                        //poussoir
    if(buttonValue == HIGH){
      counter++;                    //Incrémentation du compteur (+1)
     }
// …
}
```

Seulement le code comporte une erreur critique. La variable de compteur est incrémentée à chaque nouveau passage de la fonction `loop` quand le bouton-poussoir est enfoncé, et plus vous appuyez longtemps sur le bouton-poussoir, plus la variable est incrémentée. Mais le contenu de la variable ne doit être incrémenté que de 1 quand le bouton-poussoir est enfoncé. Comment faire pour modifier ce comportement du code ? La solution est en fait très simple. Il suffit de sauvegarder temporairement le niveau sur la broche du bouton-poussoir dans une variable après chaque interrogation. La nouvelle valeur est alors comparée à l'ancienne lors de l'interrogation suivante. Si les deux niveaux sont différents, vous devez simplement vérifier que la nouvelle valeur correspond au niveau `HIGH`, car ce sont eux qu'il faut décompter. Le nouveau niveau du moment est ensuite sauvegardé temporairement pour la prochaine comparaison, et tout reprend depuis le début.

Mais si le compteur est incrémenté à chaque appui sur le bouton-poussoir, comment fait-on pour allumer ou éteindre la LED ? Celle-ci doit pourtant s'allumer à chaque 1er, 3e, 5e, 7e appui, etc., et s'éteindre à chaque 2e, 4e, 6e, 8e appui, etc., sur le bouton-poussoir.

C'est précisément l'approche que nous avons utilisée pour résoudre le problème. Ce qu'il faut c'est évaluer d'une manière ou d'une autre le contenu de la variable du compteur. Ne remarquez-vous rien quand vous regardez les valeurs responsables de l'allumage de la LED ?

J'y suis ! Toutes les valeurs qui doivent faire allumer la LED sont impaires et les autres sont paires.

Exactement, c'est la solution. Il nous faut donc trouver le moyen de programmer quelque chose qui nous permette de tester la parité ou l'imparité d'une valeur. Je vous donne une piste : quand vous divisez un nombre par 2, vous avez un reste nul si le nombre est pair mais vous en avez un non nul si le nombre est impair. Jetons maintenant un coup d'œil au tableau 4-2.

Tableau 4-2 ▶
Division de nombres entiers par 2

Division	Résultat et reste de la division	Reste non nul ?
1/2	0 Reste 1	Oui
2/2	1 Reste 0	Non
3/2	1 Reste 1	Oui
4/2	2 Reste 0	Non
5/2	2 Reste 1	Oui
6/2	3 Reste 0	Non

On voit donc que seules des valeurs impaires donnent un reste non nul. Un opérateur spécial est utilisé en programmation pour déterminer le reste. Il s'agit de l'opérateur modulo, qui est représenté par le signe %. La première des lignes de code vérifie si la valeur du compteur est paire ou impaire :

```
if(counter % 2 == 0)     //La variable de compteur est-elle un nombre
                         //pair ?
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
```

La LED est allumée quand les valeurs sont paires, et éteinte quand elles sont impaires.

Attention !
Les opérandes de l'opérateur modulo `%` doivent être d'un type de donnée à nombre entier, tel que par exemple `int`, `byte` ou `unsigned int`.

Voyons maintenant comment le circuit se comporte quand nous appuyons plusieurs fois – disons toutes les secondes – sur le bouton-poussoir. Le résultat est présenté dans le chronogramme de la figure 4-3.

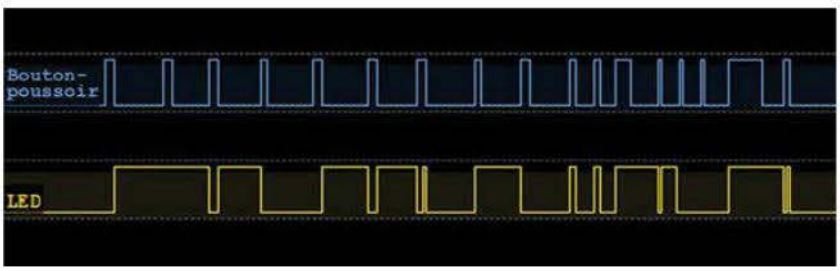


Figure 4-3
Changement du niveau de la LED pour un appui sur le bouton-poussoir

Ce n'est certainement pas le comportement que nous avions prévu. La LED ne bascule pas au rythme de l'appui sur la touche, mais a le comportement typique d'un bouton-poussoir ou d'un interrupteur à rebond. Que faire pour que le rebond n'ait pas ce genre de conséquence sur le circuit ou le compteur ? Une des solutions consiste à ajouter une temporisation pour diminuer le rebond. Ajoutez simplement un ordre `delay` derrière l'évaluation du compteur :

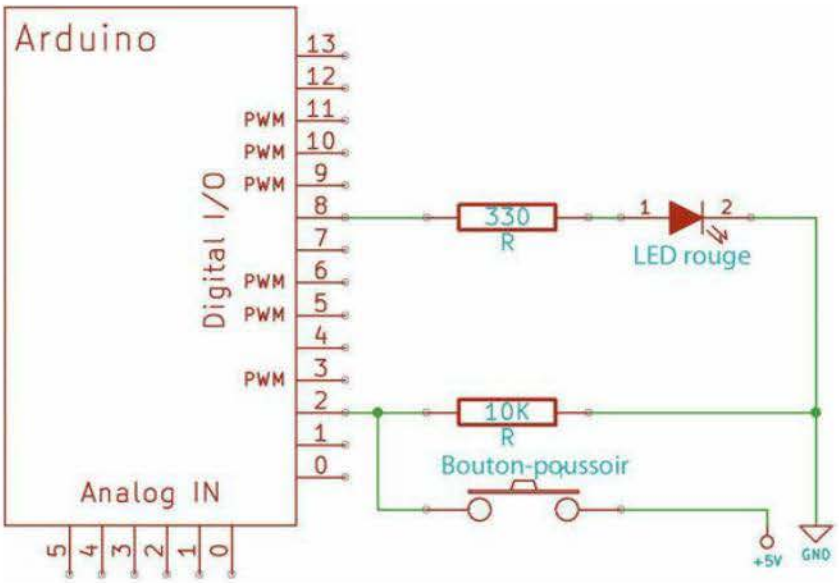
```
if(counter % 2 == 0)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
delay(10);        //Attendre 10 ms avant d'interroger à nouveau
                  //le bouton-poussoir
```

J'ai choisi ici une valeur de 10 millisecondes car elle convenait très bien pour mon bouton-poussoir. La valeur correcte ou optimale dépend naturellement toujours de la vitesse à laquelle vous souhaitez actionner le bouton-poussoir plusieurs fois d'affilée, de sorte que le logiciel puisse encore réagir. Essayez différentes valeurs et choisissez celle qui vous convient.

Schéma

Le schéma vous est certainement familier. Le logiciel utilisé est quant à lui légèrement différent.

Figure 4-4 ▶
Carte Arduino avec bouton-poussoir et LED illustrant le rebond



Autres possibilités de compenser le rebond

Nous avons jusqu'ici proposé une possibilité de compenser le rebond d'un composant mécanique tel que par exemple le bouton-poussoir. Mais il en existe d'autres :

1. boutons–poussoir spéciaux qui n'ont pas de rebond et disposent d'un point d'appui fixe ;
2. utilisation d'une bibliothèque spécialement prévue à cet effet, appelée Bounce ;
3. par l'ajout d'un petit dispositif matériel basé sur un circuit RC.

Je souhaite m'arrêter un peu sur le point 2. Si le point 3 vous intéresse également, vous trouverez de nombreuses informations sur Internet. Une bibliothèque, également appelée librairie, est un composant logiciel développé par exemple par d'autres programmeurs pour résoudre un problème particulier. Pour ne pas réinventer la roue à chaque fois, le code en question est conditionné sous forme de bibliothèque et mis à disposition des autres utilisateurs. Si ces bibliothèques sont en accès libre – et c'est la plupart du temps le cas pour l'environnement Arduino –, vous pouvez les utiliser sans problème dans votre projet.

Vous trouverez la bibliothèque Bounce sur *http://www.arduino.cc/playground/Code/Bounce*. Vous pouvez la télécharger sous forme

d'un fichier `.zip` compressé. Décompressez-le dans le répertoire `arduino-1.x.y\libraries\`, dans lequel figurent déjà d'autres bibliothèques livrées avec le logiciel Arduino. Vous devez alors obtenir une structure de fichiers comme celle de la figure 4-5.

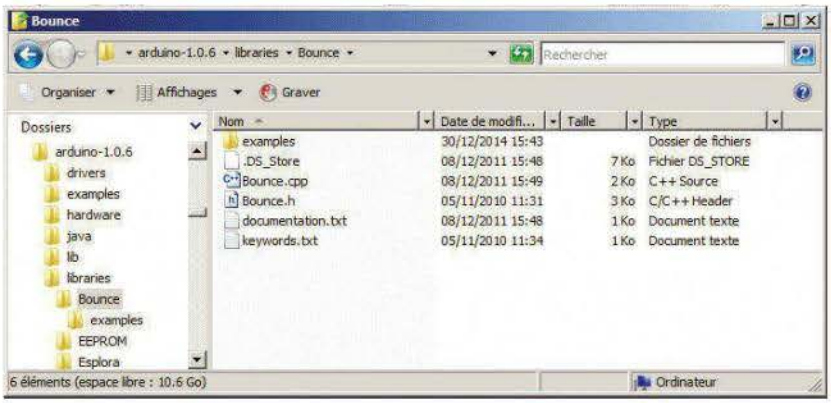


Figure 4-5
Structure de la bibliothèque Bounce

Si maintenant vous programmez le sketch dans lequel vous utilisez cette bibliothèque, vous êtes assisté par l'environnement de développement et obtenez l'aide nécessaire après avoir inséré la bibliothèque dans votre projet. Vous devez d'abord signaler d'une manière ou d'une autre à votre compilateur que vous souhaitez incorporer du code étranger. Utilisez pour cela l'instruction de prétraitement `#include`. Des explications plus précises vous seront bientôt données à ce sujet. Il vous suffit, après avoir décompressé le code dans le répertoire en question, d'ajouter l'instruction `#include` en utilisant les entrées du menu de l'IDE représentées à la figure 4-6.

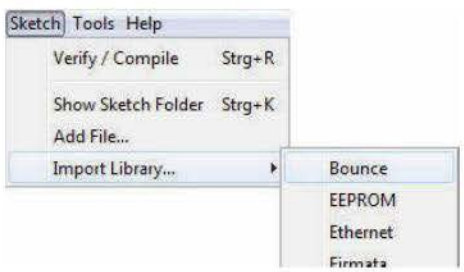


Figure 4-6
Intégration de la bibliothèque Bounce dans votre sketch

Par le menu Sketch>Import Library, on peut voir la liste de toutes les bibliothèques disponibles dans le répertoire `libraries`. On choisit l'entrée `Bounce`, ce qui écrit automatiquement la directive de préprocesseur `#include` sur la première ligne de l'éditeur. Après cette ligne, on écrit le code qui peut, par exemple, se présenter comme ce qui suit :

```
#include <Bounce.h>      //Intégrer la bibliothèque Bounce
int ledPin = 12;         //LED en broche 12
int buttonPin = 8;       //Bouton-poussoir en broche 8
int waitTime = 10;       //Temps d'attente = 10 ms
Bounce debouncing = Bounce(buttonPin, waitTime);
                         //Générer un objet Bounce
void setup(){
   pinMode(ledPin, OUTPUT);     //Broche de LED comme sortie
   pinMode(buttonPin, INPUT);   //Broche de bouton-poussoir comme entrée
}

void loop(){
   debouncing.update();           //Mise à jour de l'antirebond
   int Value = debouncing.read(); //Lecture valeur de mise à jour
   if (Value == HIGH)
     digitalWrite(ledPin, HIGH);  //Allumer LED
   else
     digitalWrite(ledPin, LOW);   //Éteindre LED
}
```

Vous saurez bientôt ce qu'est ici un objet. Prenez seulement le code comme il est. Je vous conseille d'utiliser le code que nous avons créé pour le circuit devant basculer la LED à chaque appui sur le bouton-poussoir. Il convient mieux pour vérifier que la bibliothèque Bounce fonctionne bien.

Pour aller plus loin

Pour compléter ce chapitre, vous pouvez effectuer une recherche sur Internet sur les mots-clés :

- rebond bouton-poussoir ;
- antirebond.

Réalisation du circuit

Ayant déjà construit un circuit similaire avec Fritzing (voir montage n° 2), je n'ai pas besoin de le représenter ici.

Problèmes courants

Si la LED ne s'allume pas ou ne bascule pas, plusieurs choses peuvent en être la cause.

- La LED peut avoir été mal polarisée. Rappelez-vous les deux différentes connexions d'une LED que sont l'anode et la cathode.
- La LED est peut-être défectueuse et a été grillée par une surtension lors des montages précédents. Testez-la avec une résistance série sur une source d'alimentation de 5 V.
- Vérifiez les branchements de la LED et des composants sur votre plaque d'essais.
- Vérifiez le sketch que vous avez entré dans l'éditeur de l'IDE. Peut-être avez-vous oublié une ligne ou commis une erreur ou peut-être le sketch a-t-il mal été transmis ?
- Vérifiez le bon fonctionnement du bouton-poussoir utilisé avec un testeur de continuité ou un multimètre.

Qu'avez-vous appris ?

- Vous savez maintenant que des composants mécaniques tels que des boutons–poussoir ou des interrupteurs ne se ferment ou ne s'ouvrent pas immédiatement. Plusieurs brèves interruptions successives peuvent résulter par exemple de tolérances de fabrication, d'impuretés ou de matériel en vibration, avant d'arriver à un état stable. Ce comportement est enregistré et traité comme tel par des circuits électroniques. Si par exemple vous devez compter le nombre d'appuis sur le bouton-poussoir, ces impulsions multiples peuvent s'avérer extrêmement gênantes.
- Ce comportement peut être corrigé de différentes manières :
  - par une solution logicielle (par exemple une stratégie de temporisation lors de l'interrogation du signal d'entrée) ;
  - par une solution matérielle (par exemple un circuit RC).
- Vous savez comment intégrer dans votre sketch une bibliothèque externe créée par d'autres développeurs, et aussi ce qu'est la directive de prétraitement `#include`.

Exercice complémentaire

Dans cet exercice, je voudrais que vous construisiez un circuit commandant plusieurs LED. Disons qu'il doit en avoir au moins 5. Le logiciel doit allumer une nouvelle LED dans la chaîne à chaque appui sur le bouton-poussoir. Le rebond est ainsi bien visible, quand

en effet plusieurs LED s'allument directement lors d'un appui sur le bouton-poussoir. Corrigez alors la programmation de telle sorte que le rebond n'ait plus aucune incidence, et vérifiez-la avec le circuit.

Astuce

Vous pouvez utiliser un bargraphe à LED pour composer une chaîne avec beaucoup de LED commutées l'une derrière l'autre. Il existe plusieurs versions, dans lesquelles les différentes LED sont idéalement logées dans un boîtier.

Certains composants peuvent comporter 10 et même 20 éléments de LED.

Figure 4-7 ▶
Bargraphe de type YBG 2000 avec 20 éléments de LED

Une résistance série appropriée est ici aussi impérativement nécessaire.