Montage 2

Interrogation d'un capteur

Au sommaire :

- la déclaration et l'initialisation de plusieurs variables ;
- la programmation des broches en tant qu'entrée (`INPUT`) et sortie (`OUTPUT`) ;
- l'instruction `digitalRead()` ;
- l'utilisation de la structure de contrôle `if-else` ;
- le sketch complet ;
- l'analyse du schéma ;
- la réalisation du circuit ;
- un exercice complémentaire.

Appuyez sur le bouton

Dans cet exemple, nous entendons aller à contre-courant et ne pas envoyer, comme dans notre premier sketch, des informations depuis notre carte Arduino vers l'extérieur, mais relier un composant à une broche, interroger l'état du composant et renvoyer celui-ci à une LED raccordée. Le comportement suivant doit être ici de mise :

- bouton-poussoir non enfoncé – LED éteinte ;
- bouton-poussoir enfoncé – LED allumée.

Composants nécessaires

	1 LED rouge
	1 bouton-poussoir
	1 résistance de 10 kΩ
	1 résistance de 330 Ω
	Plusieurs cavaliers flexibles de couleurs et de longueurs différentes

Code du sketch

Le code du sketch pour cet exemple est le suivant :

```
int ledPin = 13;      //LED en broche 13
int buttonPin = 8;    //Bouton-poussoir en broche 8
int buttonState;      //Variable pour enregistrer l'état du bouton
void setup(){
  pinMode(ledPin, OUTPUT);   //Broche LED en tant que sortie
  pinMode(buttonPin, INPUT); //Broche bouton-poussoir en tant
                             //qu'entrée
}

void loop(){
  buttonState = digitalRead(buttonPin);
  if(buttonState == HIGH)
     digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

Quand vous avez transmis le code, compilez-le comme vous avez appris et envoyez-le au microcontrôleur.

Pour aller plus loin

Une broche numérique opère de manière standard comme entrée et n'a donc pas besoin d'être programmée en tant que telle au moyen de l'instruction `pinMode`. C'est cependant utile pour une meilleure vue d'ensemble. Vous pouvez malgré tout laisser tomber cette étape dans la mesure où votre mémoire est juste et où chaque octet compte.

Revue de code

Dans cet exemple, on voit d'emblée qu'on a affaire à plusieurs variables qui doivent être déclarées et initialisées dès le début. Passons-le en revue.

Tableau 2-1
Variables nécessaires et leur objet

Variable	Objet
`ledPin`	Contient le numéro de broche pour la LED sur la broche de sortie numérique 13.
`buttonPin`	Contient le numéro de broche pour le bouton-poussoir sur la broche d'entrée numérique 8.
`buttonState`	Sert à enregistrer l'état du bouton-poussoir pour une exploitation ultérieure.

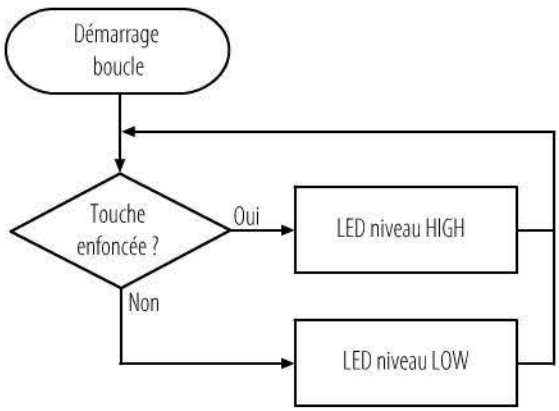


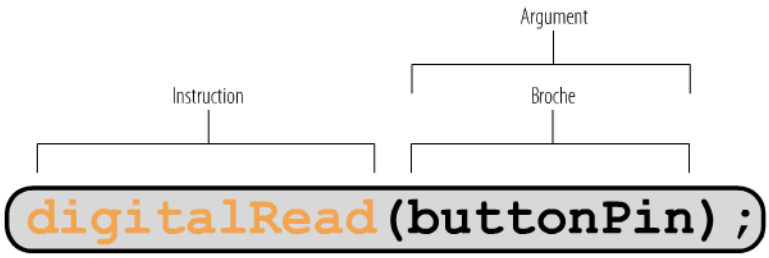
Figure 2-1
Organigramme pour commander la LED

L'organigramme se lit très facilement. Lorsque l'exécution du sketch arrive à la boucle sans fin `loop`, l'état de la broche du bouton-poussoir est continuellement interrogé et consigné dans la variable `buttonState`. Voici la ligne de code correspondante.

```
buttonState = digitalRead(buttonPin);
```

La variable est donc réinitialisée en permanence, et son comportement varie selon l'état du bouton-poussoir. La syntaxe de l'instruction `digitalRead` est indiquée dans la figure 2-2.

Figure 2-2 ▶
L'instruction `digitalRead`

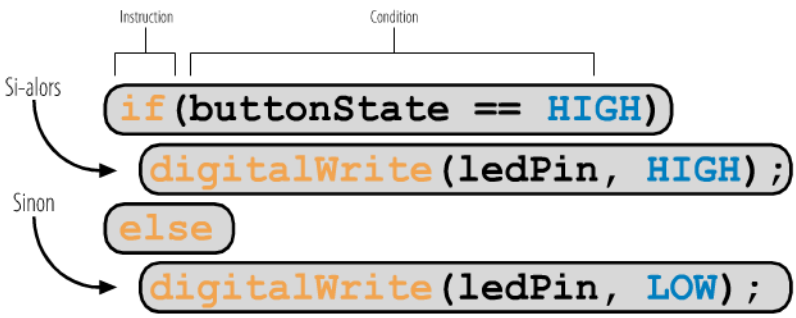


Cette fonction n'est pas seulement appelée, mais nous renvoie également une valeur de retour que nous pouvons mettre à profit. La valeur est transmise à la variable `buttonState` au moyen de l'opérateur d'affectation `=`. Les valeurs possibles peuvent être soit `HIGH`, soit `LOW` et sont ici aussi – comme vous l'avez déjà appris – des constantes qui améliorent la lisibilité. Vous savez maintenant quelles valeurs se cachent derrière, grâce au montage n° 1. L'évaluation est effectuée à la suite de l'interrogation par une structure de contrôle `if-else` (si-alors-sinon).

```
if(buttonState == HIGH)
  digitalWrite(ledPin, HIGH);
else
  digitalWrite(ledPin, LOW);
```

L'instruction `if` évalue la condition entre parenthèses, laquelle peut être librement traduite comme suit : « Le contenu de la variable `buttonState` est-il égal à `HIGH` ? Si oui, exécuter la ligne d'instruction qui vient aussitôt après l'instruction `if`. Si non, poursuivre avec l'instruction qui vient après le mot-clé `else`. »

Figure 2-3 ▶
Interrogation par structure de contrôle `if-else`



La figure 2-4 montre le mode de travail de cette structure de contrôle.

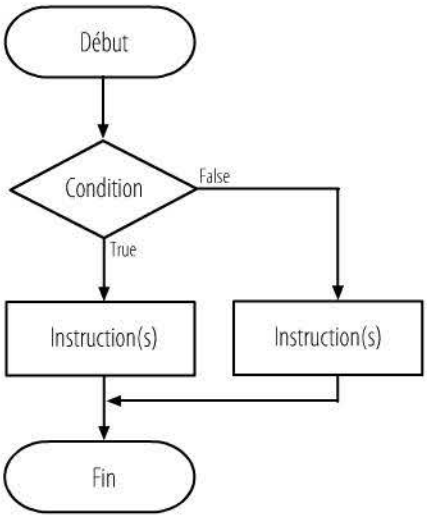


Figure 2-4
Organigramme pour structure de contrôle `if-else`

Il existe une variante plus simple de la structure de contrôle `if`, dans laquelle la branche `else` est absente. Nous y reviendrons plus tard. Vous voyez donc que le déroulement d'un programme n'est pas forcément linéaire. On peut y insérer des ramifications qui mettent diverses instructions ou blocs d'instructions à exécution au moyen de mécanismes d'évaluation. Un sketch n'agit pas seulement, mais réagit également à des influences externes, par exemple des signaux de capteur.

Attention !

Une erreur très fréquente chez les débutants consiste à confondre opérateur d'égalité et opérateur d'affectation. L'opérateur d'égalité `==` et l'opérateur d'affectation `=` ont des missions complètement différentes, mais sont souvent utilisés l'un à la place de l'autre. Le pire est que les deux modes d'écriture sont utilisables et valables dans une condition. Voici l'utilisation correcte de l'opérateur d'égalité :

```
if(buttonState == HIGH)
```

Voici maintenant l'utilisation erronée de l'opérateur d'affectation :

```
if(buttonState = HIGH)
```

Comment se fait-il que ce mode d'écriture ne produise pas des erreurs ? C'est très simple : il s'ensuit une affectation de la constante `HIGH` (valeur numérique `1`) à la variable `buttonState`. `1` n'étant pas une valeur nulle, elle est interprétée comme étant `true` (vraie). Dans le cas d'une ligne de code `if(true)...`, l'instruction qui suit est toujours exécutée. Une valeur numérique `0` est évaluée comme

`false` (fausse) dans C/C++ et toute autre différente de `0` est évaluée comme `true`.

De telles erreurs sont difficiles à discerner et cela prend toujours énormément de temps.

Schéma

Voyons d'abord le raccordement du bouton-poussoir à l'entrée numérique. Je l'ai branché sur la broche 8, de manière à ce qu'il soit légèrement éloigné de la broche 13. J'aurais bien sûr pu utiliser n'importe quelle autre broche numérique.

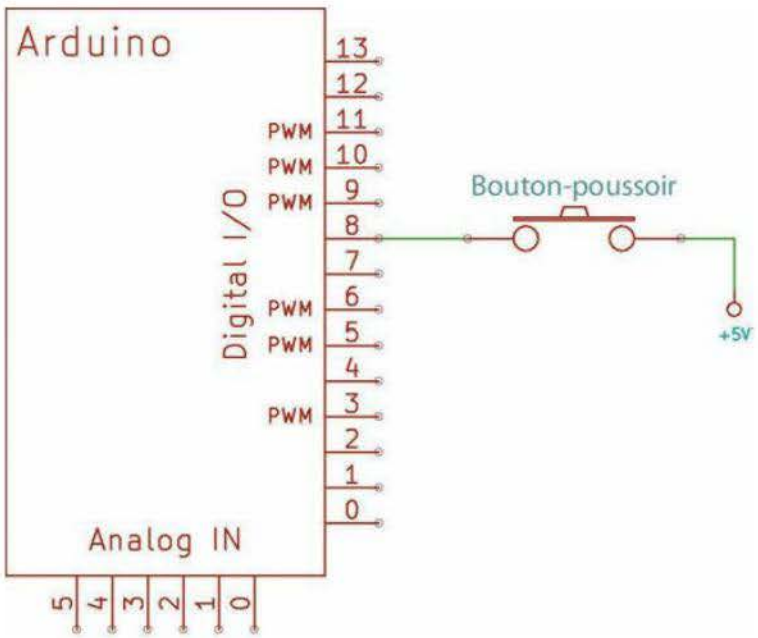


Figure 2-5 ▶
Carte Arduino avec un bouton-poussoir sur la broche 8

On voit ici que le bouton-poussoir est relié d'un côté à la broche numérique 8 et de l'autre à la tension de service de la carte Arduino, soit 5 V. C'est là tout le problème. Le circuit, tel que vous le voyez ici, ne fonctionne pas comme vous l'auriez peut-être imaginé. Si une entrée n'est alimentée par aucun niveau défini sous une forme `HIGH` ou `LOW`, le comportement dépend de facteurs divers, tels que par exemple l'énergie statique provenant de l'environnement ou l'humidité de l'air. Cela ressemble alors plus à un jeu de hasard qu'à un circuit stable. Pour remédier à ce problème, il existe diverses solutions dont certaines vous seront données au fur et à mesure. Une résistance dite *pull-down* est par exemple utilisée. Cette dernière tire littéralement le niveau ou plutôt le potentiel vers le bas. Comme un courant passe également par cette résistance, celle-ci doit être relativement forte.

Le circuit de la figure 2-6 montre cette résistance, qui tire la broche 8 à travers ses 10 kΩ (c'est la valeur empirique souvent utilisée dans la littérature) vers la masse quand le bouton-poussoir n'est pas enfoncé.

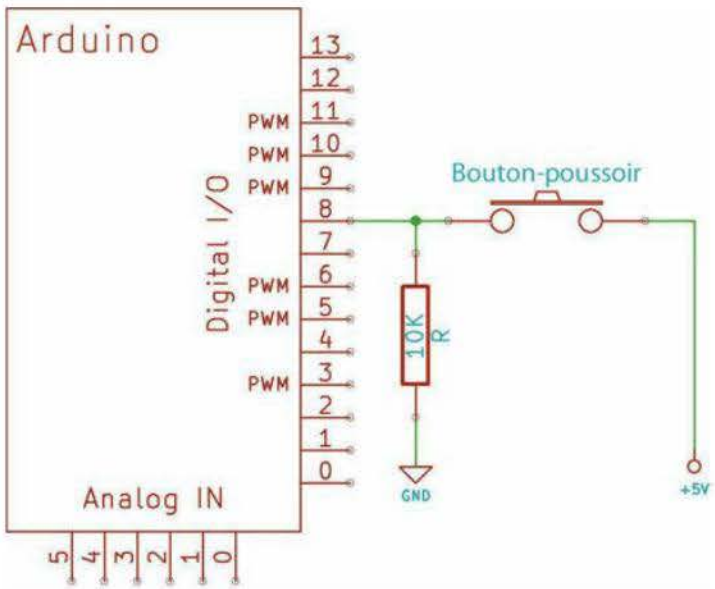


Figure 2-6
Carte Arduino avec un bouton-poussoir sur broche 8 et une résistance pull-down

Quand le bouton-poussoir est relâché, l'entrée numérique a donc un niveau `LOW` défini, clairement reconnu par le logiciel. Si, par contre, le bouton-poussoir est enfoncé, la résistance fait chuter les +5 V de la tension de service. Celle-ci est appliquée directement à la broche 8, qui est alors dotée d'un niveau `HIGH` défini. Ces connaissances préalables étant acquises, nous pouvons nous consacrer dorénavant au circuit réel.

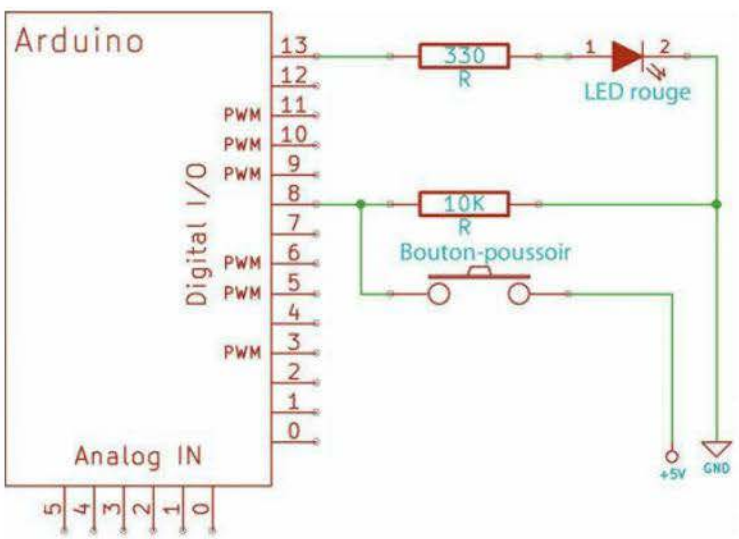
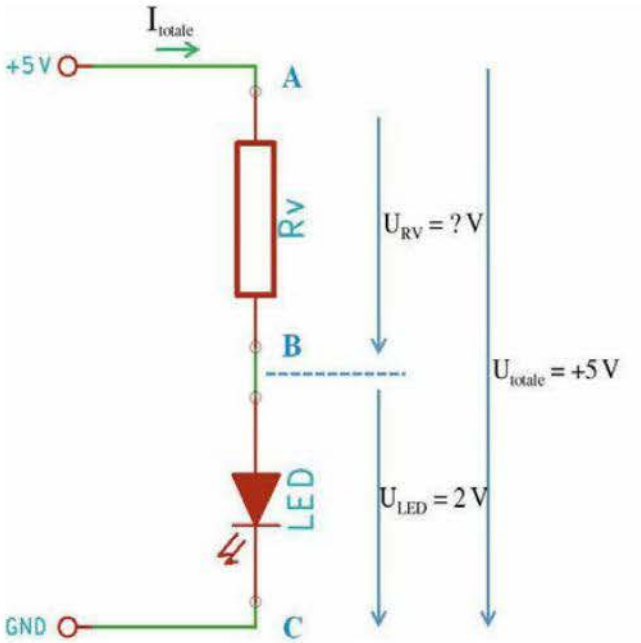


Figure 2-7
Carte Arduino avec le circuit complet pour bouton-poussoir et LED

Excusez-moi de vous interrompre à nouveau mais quelque chose me turlupine. Dans le montage n° 1, vous avez utilisé une résistance de 220 ohms comme résistance série. Celle utilisée ici fait par contre 330 ohms. Ça ressemble encore à un jeu de hasard. Que dois-je prendre au juste ?

Cette question se justifie et je me dois d'y répondre. Je vais vous montrer comment on calcule une résistance série qui marche bien et ne pose aucun problème dans un circuit. La figure 2-8 montre un circuit avec une LED et une résistance série, ainsi que les valeurs du courant et de la tension correspondantes.

Figure 2-8 ▶
LED avec résistance série et valeurs du courant et de la tension



Pour calculer la valeur d'une résistance, on utilise de nouveau la loi d'Ohm. J'ai déjà adapté la formule générale à la résistance *R* à déterminer.

$$R = \frac{U}{I}$$

Mais comment déterminer le courant et la tension ? C'est très simple : +5 V sont appliqués à la résistance et à la LED, lesquelles sont montées en série. Cette tension est délivrée par la sortie d'une broche Arduino.

Aux bornes de la LED, on a normalement entre les points B et C une chute de tension d'environ +2 V, selon la LED utilisée et sa couleur. La tension aux bornes de la résistance série – donc entre les points A et B – est par conséquent la différence entre +5 V et +2 V, soit +3 V. Reste à connaître la grandeur du courant qui passe par la résistance et la LED. Rappelez-vous que le courant passant par tous les composants électroniques est le même dans un montage en série. La fiche technique de la carte Arduino nous apprend que le courant maximal fourni par une broche est de 40 mA. Cette valeur ne doit en aucun cas être dépassée, faute de quoi le microcontrôleur peut en souffrir. C'est pourquoi nous limitons le passage du courant en insérant cette résistance série R_V dans le circuit. Il est cependant conseillé, pour plus de sécurité, de ne pas prendre 40 mA, mais une valeur légèrement inférieure. Pour calculer la résistance série, j'utilise ici deux valeurs de courant différentes soit 5 mA et 10 mA, des valeurs situées entre 5 et 30 mA sont courantes pour une LED :

$$R_1 = \frac{U_{totale} - U_{LED}}{I_1} = \frac{5\ \text{V} - 2\ \text{V}}{10\ \text{mA}} = 300\ \Omega$$

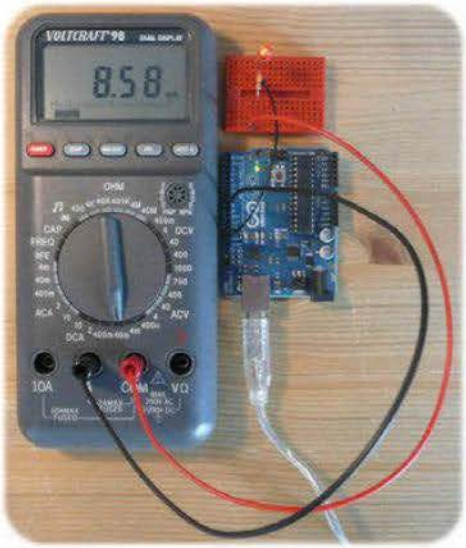
et

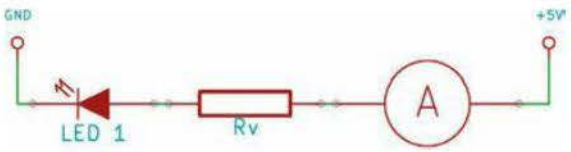
$$R_2 = \frac{U_{totale} - U_{LED}}{I_2} = \frac{5\ \text{V} - 2\ \text{V}}{5\ \text{mA}} = 600\ \Omega$$

La valeur de la résistance série doit donc être comprise entre 300 et 600 ohms pour que le port de sortie de la carte Arduino ne soit que modérément sollicité. Des valeurs de résistance plus élevées peuvent bien sûr être prises pour limiter davantage le courant, mais cela se traduirait par une baisse de luminosité pour une LED, or vous souhaitez quand-même voir encore qu'elle est allumée. J'ai choisi dans tous les autres circuits une valeur de 330 ohms pour les LED avec résistance série. Toutes les valeurs de résistance possibles ne sont pas fabriquées, mais des E-séries sont proposées avec certaines classes. Lorsque vous achetez des résistances – des assortiments pratiques sont souvent proposés –, vous devez tenir compte également de la puissance dissipée maximale. Des résistances avec une puissance dissipée d'$^1/_4$ de watt sont ici largement suffisantes. Voici la théorie. Mais il n'est pas question de mesures réelles sur l'objet actif.

J'ai branché un multimètre sur le circuit électrique de la commande de LED pour mesurer le courant.

Figure 2-9 ▶
Mesure de courant sur circuit électrique de commande de LED avec résistance série





J'ai choisi une résistance série de 330 ohms pour limiter le courant à 10 mA maxi. Le multimètre affiche un courant de 8,58 mA, soit pratiquement la valeur annoncée de 10 mA. La différence est due aux tolérances des composants et se veut même un plus faible que prévu.

Réalisation du circuit

La réalisation du circuit est déjà un peu plus complexe, c'est pourquoi nous allons nous servir de Fritzing (voir chapitre 6). Cet outil vraiment très utile est disponible sur le site Internet *http://fritzing.org*. Il va nous aider à construire le circuit et assembler les composants électroniques sur un document de travail. Vous pouvez télécharger ce logiciel gratuitement et l'utiliser pour vos projets.

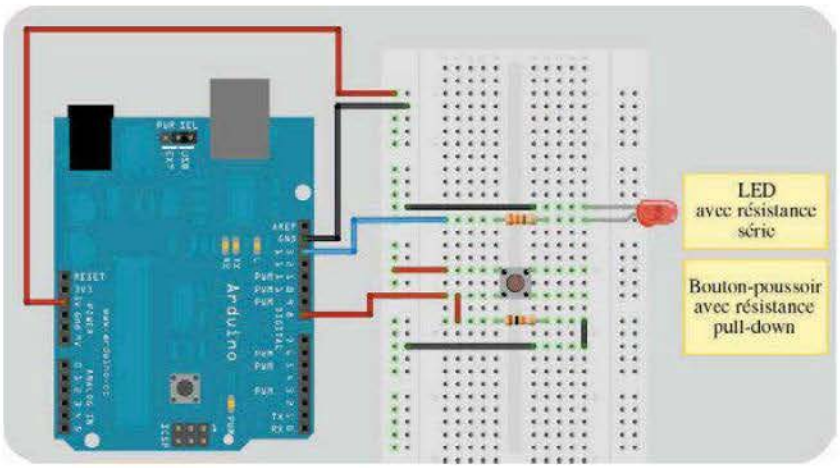


Figure 2-10
Réalisation du circuit avec Fritzing

Pour aller plus loin
Au cas où vous auriez oublié comment les différentes prises femelles d'une plaque d'essais sont reliées entre elles, consultez la section « La plaque d'essais sans soudure (breadboard) » du chapitre 7, page 155.

Problèmes courants

Si la LED ne s'allume pas quand le bouton-poussoir est enfoncé ou si la LED reste allumée, débranchez le port USB de la carte pour plus de sécurité et vérifiez ce qui suit.

- Vos fiches de raccordement sur la maquette correspondent-elles vraiment au circuit ?
- La LED a-t-elle été mise dans le bon sens, autrement dit sa polarité est-elle correcte ?
- Les boutons–poussoir peuvent être à 2 ou 4 connexions. S'il s'agit d'un modèle à 4 connexions, ont-elles été correctement branchées ? Faites, le cas échéant, un essai de continuité avec un multimètre et vérifiez ainsi l'adéquation du bouton-poussoir et des pattes correspondantes.
- Les deux résistances ont-elles bien les bonnes valeurs ou celles-ci ont-elles été interverties par mégarde ?
- Le code du sketch est-il correct ?

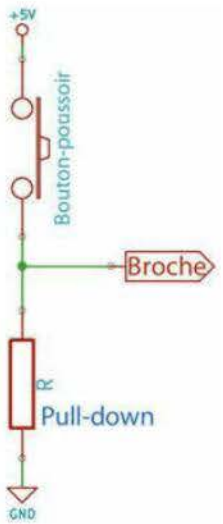
Autres possibilités pour des niveaux d'entrée définis

Avant de clore ce projet, je vais vous montrer d'autres possibilités d'avoir un niveau d'entrée défini sur une broche quand aucun signal ne lui est appliqué depuis l'extérieur. Les trois variantes suivantes sont importantes pour nous.

Avec une résistance pull-down

Vous avez déjà utilisé ce circuit.

Figure 2-11 ▶
Circuit avec une résistance pull-down



Quand le bouton-poussoir est ouvert, la broche d'entrée de votre microcontrôleur est au potentiel de la masse du fait de la résistance pull-down. Si le bouton-poussoir est fermé, la tension d'alimentation de +5 V est connectée sur la broche d'entrée.

Tableau 2-2 ▶
Potentiels de la broche

État du bouton-poussoir	Potentiel de la broche
Ouvert	0 V (masse, niveau LOW)
Fermé	+5 V (tension d'alimentation, niveau HIGH)

Tout ce qui fonctionne avec une résistance connectée à la masse peut aussi être réalisé avec une résistance connectée à la tension d'alimentation. Les potentiels sont alors exactement inversés.

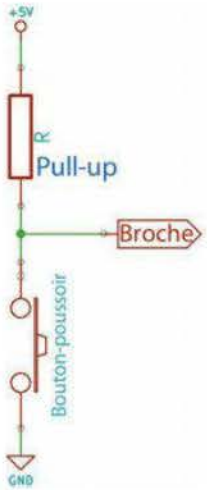


Figure 2-12
Circuit avec une résistance pull-up

Quand le bouton-poussoir est ouvert, la tension de service de +5 V est appliquée via la résistance *pull-up* à la broche d'entrée de votre microcontrôleur. Si le bouton-poussoir est fermé, la broche est immédiatement reliée au potentiel de masse.

État du bouton-poussoir	Potentiel de la broche
Ouvert	+5 V (tension d'alimentation, niveau HIGH)
Fermé	0 V (masse, niveau LOW)

Tableau 2-3
Potentiels de la broche

Avec la résistance pull-up du microcontrôleur

Une résistance pull-down ou pull-up séparée est en fait superflue car votre microcontrôleur dispose déjà de résistances pull-up internes incorporées aux broches numériques, qui peuvent être également mises en service par logiciel le cas échéant. On peut les imaginer comme suit :

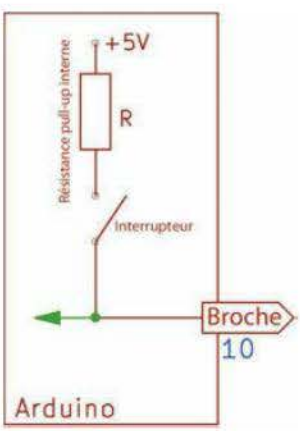


Figure 2-13
Résistance pull-up interne du microcontrôleur

J'ai choisi dans cet exemple la broche 10, à laquelle par exemple votre bouton-poussoir est relié. On peut voir que la résistance pull-up R relie la broche 10 à la tension d'alimentation de +5 V via un interrupteur électronique. La question est maintenant de savoir comment cet interrupteur peut être fermé pour que la broche présente un niveau `HIGH` en l'absence de signal d'entrée. Les instructions suivantes sont ici nécessaires :

```
pinMode(pin, INPUT);      //Programmer la broche comme entrée
digitalWrite(pin, HIGH); //Activer la résistance pull-up interne
```

Eh là, pas si vite ! Il y a quelque chose qui cloche. Vous programmez une broche en tant qu'entrée parce que vous voulez y raccorder un bouton-poussoir. Jusque-là, ça va. Mais vous envoyez quelque chose avec `digitalWrite` à cette même broche qui n'a pas été programmée en tant que sortie. Qu'est-ce que ça veut dire ?

C'est ça le truc. La séquence d'instructions en question vous permet d'activer la résistance pull-up interne de 20 kΩ, laquelle fixe le potentiel à +5 V lorsque aucun signal n'est appliqué à l'entrée.

Attention !

Si vous choisissez une des deux variantes (résistance pull-up interne ou externe), vous devrez modifier légèrement votre code. Réfléchissez un peu avant de poursuivre votre lecture. Si le bouton-poussoir est relâché, un niveau `LOW` est appliqué à l'entrée de la broche quand vous travaillez avec une résistance pull-down. Le test, pour savoir si le bouton-poussoir est enfoncé, s'effectue au moyen de la ligne suivante :

```
if(buttonState == HIGH)
```

Jusqu'ici, tout va bien. Mais si vous travaillez maintenant avec une résistance pull-up, qui génère un signal `HIGH` quand le bouton-poussoir est ouvert, vous devez écrire la ligne :

```
if(buttonState == LOW)
```

dans laquelle vous avez remplacé `HIGH` par `LOW`. Compris ?

Pour aller plus loin

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

- résistance pull-up ;
- résistance pull-down.

Qu'avez-vous appris ?

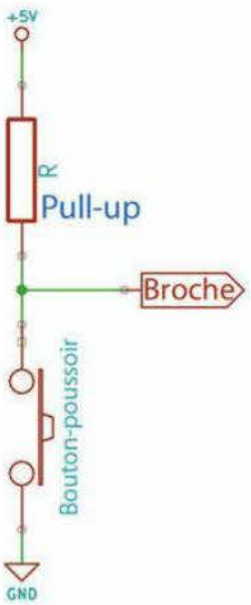
- Vous avez appris à utiliser plusieurs variables qui peuvent servir à diverses choses : déclaration pour broche d'entrée ou de sortie et enregistrement des informations d'état.
- Les broches numériques sont programmées implicitement comme entrées et n'ont pas besoin d'être explicitement programmées en tant que telles.
- Vous avez appris la fonction `digitalRead`, qui renvoie `LOW` ou `HIGH` à une entrée numérique en fonction du niveau appliqué. Cette valeur doit être affectée à une variable pour pouvoir être utilisée ultérieurement.
- Vous avez vu comment on peut contrôler le déroulement d'un sketch à l'aide de la structure de contrôle `if-else`.
- Différents schémas vous ont montré comment on représente graphiquement les connexions entre des composants électroniques.
- Une entrée numérique non connectée d'un composant électronique, dont le niveau n'est pas défini (`HIGH` ou `LOW`), conduit généralement à un comportement aléatoire et donc imprévisible du circuit.
- Aussi vous ai-je expliqué comment on utilise des résistances pull-down et pull-up, qui imposent un potentiel défini.
- Le microcontrôleur dispose de résistances pull-up internes de 20 kΩ, qui peuvent être activées via le logiciel. Vous pouvez ainsi vous épargner l'ajout de résistances pull-up externes.
- Le calcul d'une résistance série pour une LED ne vous pose maintenant plus aucun problème.
- Si vous n'avez pas lu le chapitre 6, vous avez fait la connaissance de l'outil Fritzing qui vous permet d'obtenir très rapidement des résultats dans la création de circuits par glisser-déposer.

Exercice complémentaire

Dans cet exercice, je souhaiterais vous présenter une tâche consistant à faire un *pull* de niveau numérique. Pull signifie « tirer » et c'est précisément ce que fait une résistance pull-down.

Mais le sens contraire est tout aussi possible. Une résistance pull-up permet de tirer un niveau vers le haut en direction de la tension d'alimentation. Voici à présent un tronçon de circuit que vous connaissez déjà :

Figure 2-14 ▶
Résistance pull-up



Programmez votre sketch de telle sorte que le circuit fonctionne comme indiqué. La LED s'allume quand le bouton-poussoir est enfoncé. Elle s'éteint dès qu'il ne l'est plus. Le point `Broche` dans le circuit est ici relié à la broche 8 de votre carte Arduino. La commande de la LED demeure inchangée.