# Table of Contents

# Utilisation d'un GPIO en entrée (lecture)

Les GPIO du raspberry pi peuvent être utilisés en sortie (écriture), mais également en entrée (lecture). Nous nous intéresserons ici à cette seconde possibilité, qui permettra au Raspberry pi de recevoir des informations du monde extérieur. L'objectif de ce tutoriel est de s'appuyer sur le tutoriel sur l'utilisation d'un GPIO en sortie pour contrôler une LED, et d'y ajouter le fait de lire un bouton poussoir connecté sur un autre GPIO pour changer l'état de cette LED. Il est également possible d'accéder à la liste des tutoriels sur le Raspberry pi pour voir d'autres utilisations.

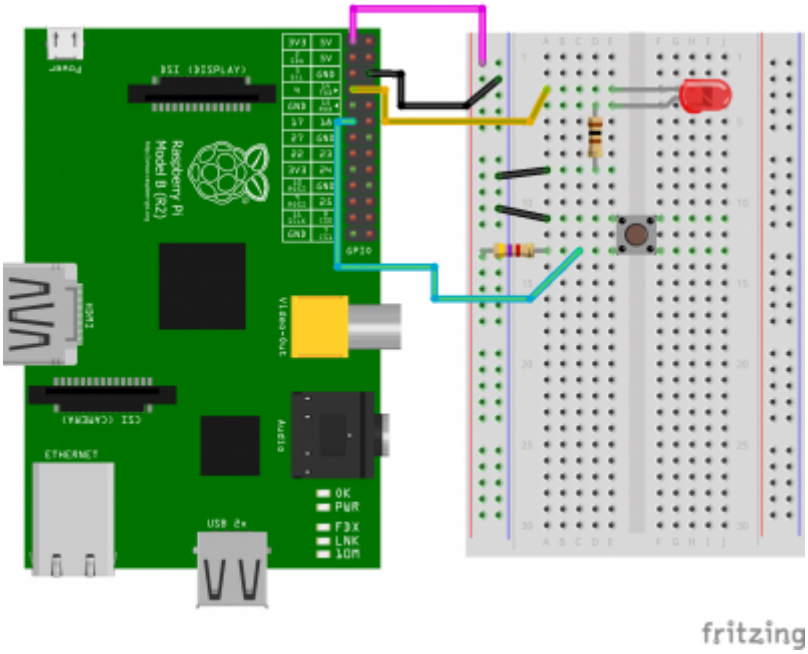
## Prérequis

Pour ce tutoriel, nous aurons besoin d'un Raspberry Pi (A, A+, B, B+) configuré, ainsi que d'une breadboard, une LED, une résistance de 68 a 300 Ohms pour la LED, d'un bouton poussoir, d'une résistance de 2KOhms à 10KOhms, et enfin de quelques câbles pour breadboard “jumper wire”. Il est également conseillé d'avoir lu le tutoriel sur l'utilisation d'un GPIO en sortie pour contrôler une LED, puisque nous nous baserons dessus.

## Montage de type "pull-up" : résistance de tirage

Nous reprendrons donc l'installation du tutoriel mentionné dans les prérequis. Nous ajouterons un câble connectant la broche en haut à gauche des GPIO au rail positif de la breadboard. Cette broche fournit du 3.3V, le niveau logique accepté par les GPIO en entrée. Il convient de faire attention à ne pas utiliser le 5V à la place, car dans ce cas, on pourrait rendre les GPIO connectés définitivement inutilisables, voir le Raspberry pi tout entier.

Le bouton poussoir sera ajouté au milieu de la breadboard, à cheval sur les deux rangées de trous. la broche du haut sera connectée à la masse, tandis qu'une résistance d'une valeur de 1 à 10 kilo-Ohms connectera la seconde broche au rail d'alimentation 3.3V. Enfin, un câble viendra s'intercaler sur entre la résistance et la broche du bouton connectée à celle ci, et sera connecté sur le GPIO 17, à savoir le sixième en partant du haut sur la colonne de gauche. Nous obtenons le résultat suivant :




.

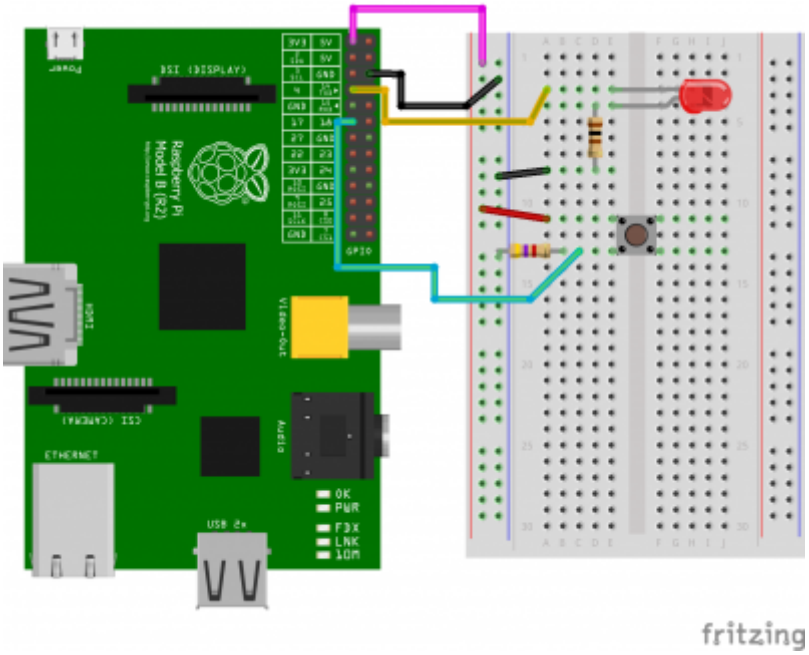
Vous pouvez également télécharger le schéma de câblage d'un bouton poussoir avec une résistance de tirage pull-up en version PDF.

Puisque nous souhaitons lire la valeur d'un bouton poussoir, nous utilisons logiquement un GPIO supplémentaire. La nouvelle résistance ajoutée ici est une résistance de tirage. Elle devra avoir une valeur comprise entre 1 et 10KOhms, ici il s'agit d'une 4.7KOhms.

Son rôle est de fixer la valeur lue lorsque l'on appuie pas sur le bouton. Sans cela, nous aurions une valeur dite “flottante”, c'est à dire variable de façon imprévisible. Dans ce cas, il s'agit d'une résistance de tirage, ou “pull-up resistor” en anglais, et elle tire la valeur vers le 3.3V, ce qui correspondra à un 1 logique (signal haut).

## Montage de type "pull-down" : résistance de rappel

Une autre solution serait de faire le contraire, c'est à dire de faire en sorte que la valeur lue par défaut, lorsque l'on n'appuie pas sur le bouton, soit nulle. Cette configuration est très proche, et utilise une résistance de rappel dite “pull down”. Notre montage change peu, puisque cette fois ci, la première broche du bouton poussoir est connectée au rail 3.3V, tandis que la résistance sera connectée à la masse, comme on peut le voir sur ce schéma :




Vous pouvez également télécharger le schéma de câblage d'un bouton poussoir avec une résistance de tirage pull-down en version PDF.

## Code minimal en C

Le code en C sera relativement simple. Ici nous donnons un exemple basé sur une résistance de tirage (pull-up), mais il suffira d'inverser les valeurs logiques pour avoir le code pour une version rappel (pull-down).

readPullUp.c

```
#include <stdio.h>
#include <wiringPi.h>
int main(void)
{
int switchPin=0;
if(wiringPiSetup()==-1)
      {return 0;}
//le port GPIO du bouton est configuré en lecture
pinMode(switchPin,INPUT);
int button=0;
while(1)
{
  //on lit la valeur de la broche GPIO
  button=digitalRead(switchPin);
  if(button==0)//Si un appui sur le bouton est détecté
  {
   //on affiche un message
   printf("button pressed!\n");
   //cette boucle permet de gerer un appui continu
   while(button==0)
    {
            //on relit la valeur à chaque fois
```

```
            button=digitalRead(switchPin);
            delay(20);//et on attend 20ms
    }
  }
  delay(20);//on attend 20ms entre chaque lecture.
}
  return 0;
}
```

On compile ensuite le code :

```
gcc readButton.c -o readButton -lwiringPi
```

On peut alors exécuter le programme :

```
sudo ./readButton
```

L'attente de 20 millisecondes (*delay(20);*) sert non seulement à ne pas surcharger le CPU, mais également à filtrer le rebond du bouton. Comme il s'agit d'un dispositif imparfait, sans cette attente, nous aurions une successions de changements d'états (0-1-0-1-0-1 ...) lors des phases d'appui et de relâchement du bouton. Ce phénomène ne survient pas lorsque le bouton est maintenu enfoncé ou lorsque l'on ne le touche pas, mais uniquement sur les transitions. La seconde boucle permet de ne considérer qu'un seul appui tant que le bouton est maintenu enfoncé, sinon, nous aurions une succession d'appuis, et un affichage toutes les 20ms tant que le bouton est maintenu enfoncé. Cela peut paraître superflu, mais sans cela, si vous appuyez fugitivement sur le bouton, mais tout de même plus de 20ms, il y aurait au moins deux pressions de comptabilisées.

Cela s'appelle en anglais le "*debouncing*". On peut également utiliser un condensateur pour atténuer le rebond, et potentiellement ne pas avoir à faire de debouncing logiciel.

## Code utilisant le bouton et la LED en C

On peut maintenant réutiliser la LED installée pour cette fois ci changer l'état de celle ci selon les appuis. On obtient un code de ce genre :

ledControlWithButton.c

```
#include <stdio.h>
#include <wiringPi.h>
int main(void)
{
int switchPin=0;
int ledPin =7;
if(wiringPiSetup()==-1)
          {return 0;}

//le port GPIO du bouton est configuré en lecture
```

```
pinMode(switchPin,INPUT);
//le port GPIO de la LED est configuré en ecriture
pinMode(ledPin,OUTPUT);
int button=0;
int ledState=0;//état initial de la LED
digitalWrite(ledPin,ledState);//on eteint la LED au départ
while(1)
{
  //on lit la valeur de la broche GPIO
  button=digitalRead(switchPin);
  if(button==0)//Si un appui sur le bouton est détecté
  {
   //on affiche un message
   printf("button pressed!\n");
   if(ledState==0)
       {ledState=1;}
   else
       {ledState=0;}
   digitalWrite(ledPin,ledState);//on applique le nouvel état de la LED
   //cette boucle permet de gerer un appui continu
   while(button==0)
   {
       //on relit la valeur à chaque fois
       button=digitalRead(switchPin);
       delay(20);//et on attend 20ms
   }
  }
  delay(20);//on attend 20ms entre chaque lecture.
}
  return 0;
}
```

## Code minimal en python

Voyons maintenant le code minimal en python :

readButton.py

```
import time
from RPi import GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(0, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
while True:
    inputval = GPIO.input(0)
    print inputval
    time.sleep(1)
```

On exécute le code par :

```
sudo python readButton.py
```

# Conclusions

Il est donc possible d'utiliser un GPIO comme entrée, pour lire des informations du monde extérieur. Ici, nous lisons des données numériques. Il est toutefois possible d'utiliser un convertisseur MCP3008 pour ajouter des entrées analogiques au Raspberry pi et ainsi lire des données analogiques.

Il est primordial de ne pas envoyer de courant à une tension supérieure à 3.3V aux GPIO sous peine de risquer d'endommager le/les GPIO voir le Raspberry pi tout entier.

On pourra se demander comment avons nous choisi la valeur de la résistance de tirage/rappel. Il s'agit en fait de valeurs assez universelles dans ce types de montages. Si l'on ne sait pas quoi prendre, les 4.7 et 10K feront généralement l'affaire. Il est cependant possible d'aller plus loin en cherchant les valeurs optimales selon le circuit [1)]

1)

Comment choisir la valeur d'une résistance de tirage/rappel? - (en) http://electronics.stackexchange.com/questions/23645/how-do-i-calculate-the-required-value-for-a-pull-up-resistor